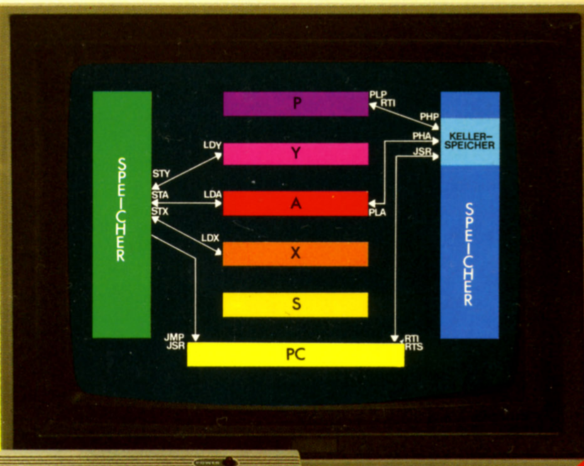


Band 4:
Ein Leitfaden für Systemprogrammierer



Hans Lorenz Schneider • Werner Eberl

Das Commodore 64 Buch





Assembler
Disassembler

Das Commodore 64-Buch
Band 4

Hans Lorenz Schneider
Werner Eberl

Das Commodore 64-Buch

Band 4: Ein Leitfaden für Systemprogrammierer

Markt & Technik Verlag

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Schneider, Hans Lorenz:
[Das Commodore-vierundsechzig-Buch]
Das Commodore-64-Buch / Hans Lorenz Schneider ;
Werner Eberl. — Haar bei München : Markt-und-Technik-Verlag
(Computer persönlich)
NE: Eberl, Werner:
Bd. 4. Ein Leitfaden für Systemprogrammierer. — 1984.
ISBN 3-922120-70-9

ISBN 3-922120-70-9

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Eimannsberger, München
Printed in Germany

Vorwort

Programmieren in Assembler, Programmieren in Maschinensprache, das sind Zauberwörter für jeden Home-Computer Besitzer. Zauberwörter zum einen, weil erst durch diese Art der Programmierung die Fähigkeiten des Home-Computers voll ausgeschöpft werden können. Zauberwörter aber auch, weil die Maschinensprache zunächst unverständlich und geheimnisvoll erscheint.

Was aber nützt einem da ein fertig gekaufter Assembler, wenn dabei nicht erklärt ist, was man damit machen kann. Wenn man es schließlich gelernt hat, in Assembler zu programmieren, taucht die zweite Schwierigkeit auf: Man möchte die erstellten Programme mit Basic zusammen arbeiten lassen. Ein ROM-Listing des Betriebssystems bietet aber nur ungenügend Aufschluß über die Funktion des gesamten Systems.

Deshalb wollten wir in diesem Buch einen Leitfaden vorstellen, mit dem Sie die Programmierung in Assembler auf Ihrem Commodore 64 mit Hilfe von immer wieder eingefügten Beispielen erlernen können. Schließlich werden noch Tips gegeben, wie man die vorgestellten Programme verändern kann, als Übung für Sie.

Wir wünschen Ihnen viel Erfolg, wenn Sie Ihre Ideen mit dem neuen Werkzeug des Assemblers in lauffähige Programme umsetzen.

München im März 1984

Hans Lorenz Schneider **Werner Eberl**

Einleitung

Wir haben versucht, das Buch so aufzubauen, daß Sie keine weitere Literatur zum Verständnis der Assembler-Programmierung auf Ihrem Commodore 64 benötigen. Deshalb erläutern wir im ersten Kapitel zunächst, was Programmieren in Maschinensprache oder Assembler bedeutet, und welche Vorteile und Nachteile daraus entstehen. Dann werden die Befehle des eingebauten Prozessors (6510) vorgestellt und genau beschrieben. In diesem Kapitel finden Sie auch bereits einfache Beispiele, und schon hier sollten Sie in der Lage sein, einfache Probleme in Maschinensprache zu formulieren.

Kapitel 2 beschreibt nun die Möglichkeiten, ein vorhandenes Maschinenprogramm mit Basic zu verknüpfen. Nach Erläuterung einiger Grundlagen über Speicheraufbau und Zahldarstellungen gehen wir sofort auf die im Commodore 64 eingebauten ROM-Routinen ein, sowie auf die Möglichkeit ihres Aufrufes.

In Kapitel 3 wird ein Assembler beschrieben, der vollständig in Basic geschrieben ist und dadurch auch leicht modifiziert werden kann. Weil dieser Assembler jedoch recht langsam arbeitet, beschreiben wir im nächsten Kapitel Möglichkeiten, häufig verwendete Unterprogramme des Assemblers selbst in Assembler zu formulieren. Dies ist gleichzeitig ein sehr lehrreiches Beispiel zur Anwendung der Assembler-Programmierung im Zusammenhang mit einem Basic-Programm. Viele der dort vorgestellten Routinen sind auch für andere Basic-Programme nützlich und sie werden später sicher oft in diesem Kapitel Anregungen für Ihre eigenen Anwendungen finden.

Kapitel 5 beschreibt einen komfortablen Disassembler, mit dem es möglich ist, fast den gesamten Quellcode eines Assemblerprogramms wiederzugewinnen.

Als Zusammenfassung oder für jemand, der die Diskette zum Buch kauft, sind in Kapitel 6 die Bedienungsanleitungen für Assembler und Disassembler abgebildet.

Um Ihnen einen ersten Überblick über die Leistung des Assemblers zu geben, hier eine kurze Aufstellung seiner wichtigsten Merkmale:

- läuft auf Commodore 64 mit Floppy 1541
- Ein-Pass-Assembler
- Quelldatei kann als Programmdatei oder sequentielle Datei vorliegen
- die Quelldatei kann beliebig groß sein
- die Objektdatei wird als Programmdatei gespeichert
- Vorwärtsverweise werden gekennzeichnet und automatisch eingesetzt
- Verwendung von Symbolen für Konstanten oder Marken ist möglich
- Einfache Arithmetik in Ausdrücken ist zulässig
- bis zu 255 Symbole und bis zu 200 Vorwärtsverweise können gespeichert werden
- einfache Handhabung mit Direktiven
- Ausgabe des Protokolls auf Bildschirm, Drucker oder Floppydatei
- durch Verwendung von Maschinenprogrammen werden kurze Assemblierungszeiten erreicht
- Einbinden von Quelldateien

Das COMMODORE 64 - BUCH

Band 4 : Leitfaden für den System-Programmmierer

Inhaltsverzeichnis

1

6510 — der Prozessor des Commodore 64

1. 6510 - Der Prozessor des Commodore 64

Oft kann man immer wieder lesen - vor allem in popular-wissenschaftlichen Artikeln oder Werbeprospekten von Computer-Firmen - wie etwa: "Dieser Microprozessor übernimmt die Aufgabe von über 150.000 Transistoren", "der Computer versteht nur '0' und '1' "," dieser neue 16-Bit-Prozessor ..." . Diese Schlagworte klingen sehr nach Elektronik, also zunächst gar nicht nach dem was, wir vom Computer gewohnt sind, nämlich Rechnen mit Zahlen und Verwalten von Daten. Es ist auch notwendig, sich lange Zeit mit Rechnern zu beschäftigen, um den Zusammenhang dieser verschiedenen Gebiete zu verstehen. Wir wollen in diesem Buch möglichst viel zum Verständnis der internen Rechnervorgänge beitragen, auch wenn eine alles umfassende Darstellung in solchem Umfang nicht gegeben sein kann.

Doch nun zum Kernstück eines Rechners: dem Prozessor. Der Prozessor im Commodore 64 nennt sich 6510 und ist ein 8-Bit-Prozessor. Der 6510 ist voll softwarekompatibel mit dem weit verbreiteten 6502. Deshalb ist das folgende Kapitel auch für den 6502 und alle analogen Prozessoren dieser Familie (6503, 6504, 6505, 6506, 6507, 6512, 6513, 6514, 6515, 65C02 / allgemein bezeichnet als 65xx) gültig.

Wir wollen nach einigen Bemerkungen über die Funktionsweise eines Mikroprozessors auf die Register, die Flags und auf die möglichen Adressierungsarten des 6510 eingehen, sowie anschließend die Befehle in alphabetischer Reihenfolge darstellen.

Dann sollen die Anwendung der Befehle an einigen einfachen Beispielen aufgezeigt werden. Zum Abschluß bringen wir noch eine Tabelle mit den Befehlen in hexadezimaler Reihenfolge.

1.1. Funktionsweise eines Mikroprozessors

Einer der wichtigsten Bausteine im Computer ist der Prozessor, auch CPU genannt. Sehen wir uns zunächst die Anschlußbelegung des 6510 an, wie er im Commdore 64 eingebaut ist.

Anschlußbelegung

CPU 6510

Signal	Pin	Pin	Signal
PHI 1	1	40	$\overline{RES}$
RDY	2	39	PHI 2
$\overline{IRQ}$	3	38	$R/\overline{W}$
$\overline{NMI}$	4	37	D_0
AEC	5	36	D_1
+5 V	6	35	D_2
A_0	7	34	D_3
A_1	8	33	D_4
A_2	9	32	D_5
A_3	10	31	D_6
A_4	11	30	D_7
A_5	12	29	P_0
A_6	13	28	P_1
A_7	14	27	P_2
A_8	15	26	P_3
A_9	16	25	P_4
A_{10}	17	24	P_5
A_{11}	18	23	A_{15}
A_{12}	19	22	A_{14}
A_{13}	20	21	GND (0V)

Bild 1.1.1 : Anschlußbelegung des 6510

Diese Anschlüsse können wir nach ihrer Bedeutung einteilen: in Versorgungsanschlüsse (GND-0V-Masse, +5V-Stromversorgung, PHI1-Takteingang, PHI2-Takteingang), sogenannte Adressleitungen (A0 - A15 / 16 Stück), Datenleitungen (D0 - D7 / 8 Stück), sowie Steuerleitungen (RES-Reset, IRQ-Interrupt Request/anfordern Unterbrechung, NMI-non-maskable-interrupt/nicht maskierbarer Interrupt, RDY-Speicherfertigmeldung, AEC-adressbus-enable-control/Steuerauswahl für Adressbus, R/W-Schreib/Lese-Leitung); außerdem sind am 6510 noch 6 Ein-/Ausgabeanschlüsse (P0 - P5) vorhanden. Diesen Zusammenhang können wir uns grafisch wie folgt veranschaulichen:

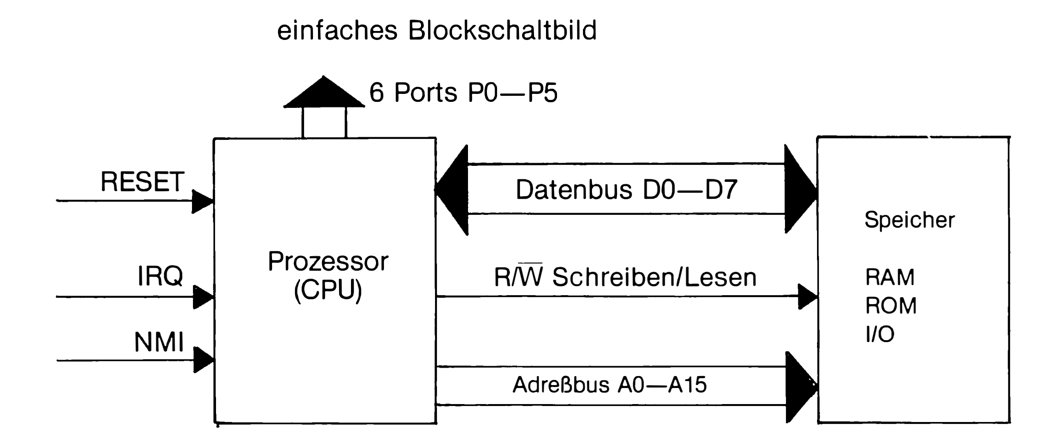


Bild 1.1.2 : Einfaches Blockschaltbild des 6510

In diesem Bild sind alle Leitungen weggelassen, die für das Verständnis im folgenden keine Bedeutung haben.

Der eingezeichnete Speicher ist eine Vorrichtung, die es erlaubt Informationen (prinzipiell als einzelne Bits '0' oder '1') zu speichern und wieder zu lesen. Logisch zusammengefaßt werden immer acht Bits zu einem Byte. Jedes Byte erhält eine Adresse, also eine Nummer unter der es angesprochen werden kann. Bei 64K (= 65536) Bytes können dies Werte von 0 bis 65535 sein. Welche Speicherzelle (Byte) gelesen oder geschrieben werden soll, bestimmt in diesem einfachen Modell alleine der Prozessor. Ob geschrieben oder gelesen werden soll, wird über die Leitung R/W dem Speicher mitgeteilt. Diese 8-Bit liegen dann paralell auf den Datenbus an, dabei entspricht dem logischen Wert '0' die Spannung 0 Volt, und 5 Volt entspricht '1'. Ähnlich wie man die Bitkombination auf dem Adressbus als Speicherzellennummer auffaßt, so faßt man die Bitkombination am Datenbus als 8-Bit-Binärzahl (Dezimalwerte 0 bis 255) auf.

Was geschieht nun beim Einschalten des Systems? Hier wird der Prozessor durch einen kurzen Impuls an der RESET-Leitung in einen definierten Ausgangszustand versetzt. Dann geschieht folgendes: Der Prozessor adressiert im Speicher die Zelle 65532, liest diesen Wert und merkt ihn sich zur weiteren Verarbeitung. Dann wird die Zelle 65533 gelesen und der erhaltene Wert mit 256 multipliziert und zum vorherigen hinzuaddiert. Das erhaltene Ergebnis ist nun die Adresse des ersten auszuführenden Befehls. Im Prozessor

ist ein 16-Bit-Register enthalten, das den momentanen Wert der Befehlsadresse speichert, ein sogenannter Programmzähler (englisch: Program Counter, abgekürzt: PC). Der Prozessor liest also nun den Inhalt der über den Programmzähler adressierten Speicherzelle. Diesen Wert interpretiert er als Befehl. Bei dieser Interpretation wird noch unterschieden nach Befehlen, die sofort ausgeführt werden können und solchen, bei denen noch ein oder zwei Byte gelesen werden müssen, die den Befehl näher beschreiben; z.B. muß bei einem Sprungbefehl noch das Sprungziel eingelesen werden.

Wenn wir die Aufgabe hätten, ein Computersystem zu entwerfen und dafür die notwendigen Maschinenprogramme zu schreiben, so müßten wir also im Prinzip dafür sorgen, daß in den Speicherzellen 65532 und 65533 die Anfangsadresse des Programmes steht und der Prozessor anschließend immer die gewünschten Werte im Speicher vorfindet, die er dann als Befehl interpretieren kann. Und das bedeutet nichts anderes, als den Prozessor zu programmieren. Die Ausdrucksweise im vorigen Absatz mag etwas umständlich erscheinen, jedoch wird dadurch die komplizierte Funktionsweise eines Prozessors besser deutlich.

Wir können also ein Programm schreiben, indem wir einfach eine Liste von Werten anfertigen, die der Prozessor nacheinander zu lesen hat. Diese Liste kann auch direkt mit Hilfe des BASIC-Befehls POKE eingegeben werden, wenn es sich um extrem kurze Programme handelt. Wenn wir jedoch größere Programme schreiben wollen, so müssen wir von den Zahlenkombinationen übergehen zu sogenannten mnemotechnischen Bezeichnungen, deren Buchstaben mehr auf die eigentliche Funktion des Befehls hinweisen, so z.B. der Befehl JMP (von englisch jump = springen), der einfach den Programmzähler mit den dem Befehlscode folgenden zwei Bytes besetzt; diesen Vorgang bezeichnen wir normalerweise als Sprung.

Wir hoffen, daß der Zusammenhang zwischen der logischen Operation 'Sprung' und den Impulsen am Prozessor selbst einigermaßen deutlich geworden ist. Es ist auch eine faszinierende Erkenntnis, in wieviel elektronische Schritte ein einfacher Programmbefehl zerteilt werden muß, damit er vom Rechner ausgeführt werden kann.

1.2 Die Register des 6510

Der 6510 enthält acht Register, davon fünf 8-Bit-Register für Arithmetik und Adressierung, ein 16-Bit-Register für

den Programmzähler und zwei 8-Bit-Register für die Ein-/ Ausgabe Kanäle. Für die letzten beiden gibt es keine spezifischen Befehle, sondern sie werden wie Speicherzellen behandelt. Hier eine Abbildung der Datenflüsse im 6510 zwischen den sechs wichtigsten Registern:

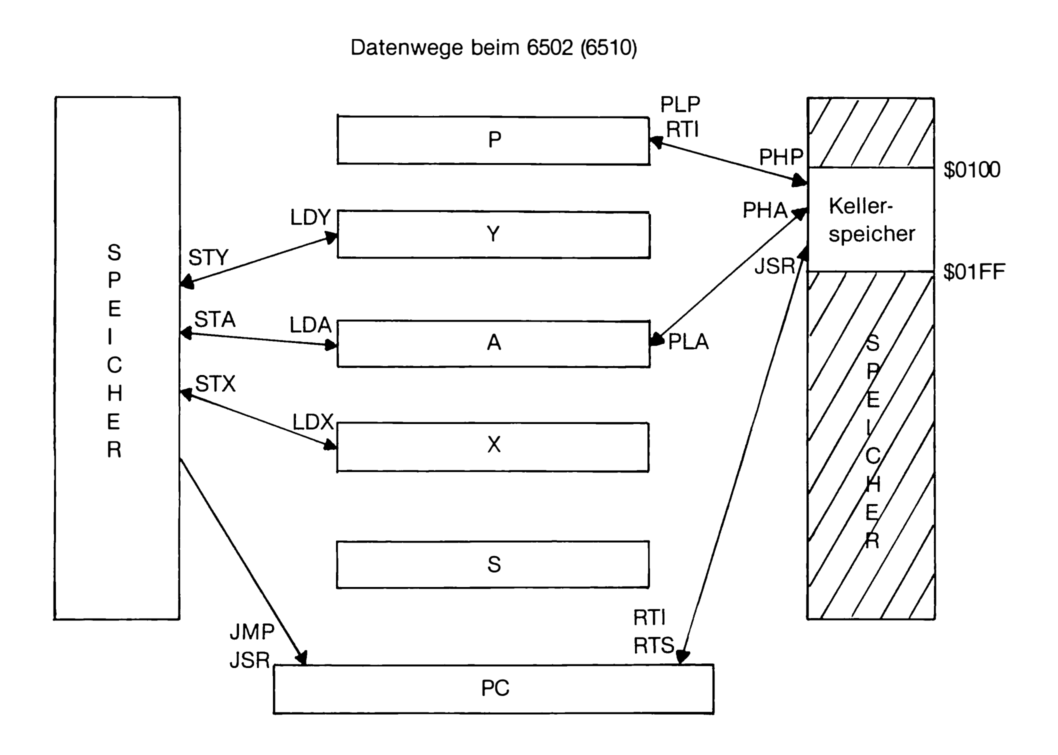


Bild 1.2 : Datenwege beim 6510

Der Programmzähler (PC)

Der Programmzähler enthält jeweils die Adresse des nächsten zu holenden Befehles. Nach jedem Befehl wird er entsprechend erhöht oder - bei einem Sprungbefehl - neu geladen. Sprungbefehle können sein: Ein direkter oder indirekter Sprung (JMP), ein Unterprogrammaufruf (JSR), eine bedingte Verzweigung (z.B. BNE) sowie Rückkehrbefehle aus

Unterprogrammen (RTS) oder Unterbrechungen (RTI). Auch durch einen hardwaremäßig ausgelösten Interrupt (auf die Theorie der Interrupts können wir hier leider aus Platzmangel nicht weiter eingehen) oder ein BRK-Befehl verändern natürlich den Programmzähler.

Akkumulator (A)

Der Akkumulator ist wohl das wichtigste Register des Prozessors. Mit ihm allein können arithmetische Operationen ausgeführt werden. Der Akkumulator kann direkt mit dem Wert einer Speicherzelle geladen werden, und er kann auch direkt in eine Speicherzelle abgelegt werden.

X-Register (X)

Dieses Register dient vor allem zur Indizierung von Tabellen (vergleiche Kapitel 1.4 und 1.6) sowie als Laufvariable in Schleifen. Als arithmetische Operationen sind hier nur INX (erhöhe X-Register um 1) und DEX (vermindere X-Register um 1) möglich.

Y-Register (Y)

Das Y-Register arbeitet wie das X-Register als Indexregister oder Laufvariable für Schleifen, und ist entsprechend durch die Befehle INY und DEY ansprechbar.

Prozessor-Status-Register (P)

Im Prozessor-Status-Register sind alle Flags (Merker) zusammengefaßt. Die Bedeutung der einzelnen Flags ist in Kapitel 1.3 beschrieben.

Stack-Pointer / Kellerspeicher (S)

Im Speicher ist der Bereich von $0100 bis $01FF besonders ausgezeichnet. Dieser Bereich arbeitet in einem 6510-System als Kellerspeicher oder Stapel (engl.: Stack). Das Register S zeigt dabei immer auch den nächsten zur Verfügung stehenden Platz im Kellerspeicher. Da dieses Register nur 8 Bit umfaßt, kann der Kellerspeicher auch nur 256 Byte groß sein.

Die Bearbeitung des Kellerspeichers geschieht mit sogenannten Push- und Pull-Operationen und dient zum kurzzeitigen

Zwischenspeichern von Werten. Nehmen wir als einfachstes Beispiel das Pushen (PHA) und Pullen (PLA) des Akkumulators. Der PHA-Befehl legt den Wert des Akkumulators an der über den Stackpointer beschriebenen Adresse im Kellerspeicher ab und vermindert den Stackpointer um eins. Damit ist der Akku im Kellerspeicher abgelegt. Das Zurückholen geht in umgekehrter Reihenfolge, d.h. beim PLA-Befehl wird zunächst der Stackpointer um eins erhöht und dann der Akkumulator mit der über den Stackpointer adressierten Speicherzelle geladen. Anders ausgedrückt arbeitet der Kellerspeicher nach dem LIFO-Prinzip (Last in, First out).

In der oben abgebildeten Grafik (Bild 1.2) sind alle Befehle eingetragen, die den Stackpointer beeinflussen.

Datenrichtungsregister

Der 6510 besitzt sechs Ein- / Ausgabeanschlüsse (P0 bis P5) deren Richtung (Ein- oder Ausgabe) durch dieses Register bestimmt wird. Eine '1' im entsprechenden Bit bedeutet Ausgang, eine '0' Eingang. Es sind hier nur die sechs niederwertigen Bits veränderbar. Die Anschlüsse, die den Ports P6 und P7 entsprechen, werden intern für die Anschlüsse NMI und RDY verwendet, und sollten deshalb ständig auf Eingang geschaltet bleiben. Die Bedeutung der einzelnen Ports im Commodore 64 ist im Kapitel 2.1 beschrieben. Das Datenrichtungsregister wird durch die Speicheradresse $0000 angesprochen.

Ein- / Augsgabe-Register

Im Ein- / Ausgabe-Register werden die Daten, die über die Ports P0-P5 gelesen oder geschrieben werden, übergeben. Dieses Register wird durch eine Lese- oder Schreiboperation mit der Adresse #0001 angesprochen.

1.3 Die Flags des 6510

Ein Flag (oder auch Flagge) ist ein Bit, das einen bestimmten Zustand des Prozessors anzeigt und wird in deutsch auch Merker genannt. Der 6510 kennt sieben verschiedene Flags. Die meisten Befehle arbeiten irgendwie in Abhängigkeit von der Stellung der Flags oder verändern diese selbst. Deshalb ist im folgenden bei jedem Flag eine Liste der Befehle angegeben, die durch das entsprechende Flag beeinflußt werden, außerdem sind die Befehle aufgeführt, die das jeweilige Flag beeinflussen. Vergleiche

auch mit der Befehlstabelle in Kapitel 1.5.

Das Carry-Flag (Übertrags-Bit)

Seinen Namen hat das Carry-Flag von seiner Bedeutung beim Addieren und Subtrahieren. Da der Akkumulator nur acht Bits umfaßt, tritt bei einer Addition, dessen Ergebnis größer als 255 ist, ein Übertrag auf. Dieser Fall wird mit dem Carry-Flag angezeigt. Außer bei der Addition können auch bei Schiebebefehlen Überträge auftauchen, die dann ebenfalls im Carry-Bit gespeichert werden. Das Carry-Bit hat noch weitere Bedeutungen, diese wollen Sie aber bitte bei den einzelnen Befehlen nachschlagen.

Befehle, die das Carry-Flag beeinflussen: ADC, ASL, CLC, CMP, CPX, CPY, LSR, PLP, ROL, ROR, RTI, SBC, SEC.

Befehle, die durch das Carry-Flag beeinflußt werden: ADC, BCC, BCS, PHP, ROL, ROR, SBC.

Zero-Flag (Ergebnis ist 0)

Dieses Bit wird immer dann gesetzt, wenn das Ergebnis einer arithmetischen Operation oder einer Ladeoperation gleich Null ist, d.h. daß anschließend im geladenen Register nur Nullen vorhanden sind.

Befehle, die das Zero-Flag beeinflussen: ADC, AND, ASL, BIT, CMP, CPX, CPY, DEC, DEX, DEY, EOR, INC, INX, INY, LDA, LDX, LDY, LSR, ORA, PLA, PLP, ROL, ROR, RTI, SPC, TAX, TAY, TYA, TSX, TXA.

Befehle, die durch das Zero-Flag beeinflußt werden: BEQ, BNE, PHP.

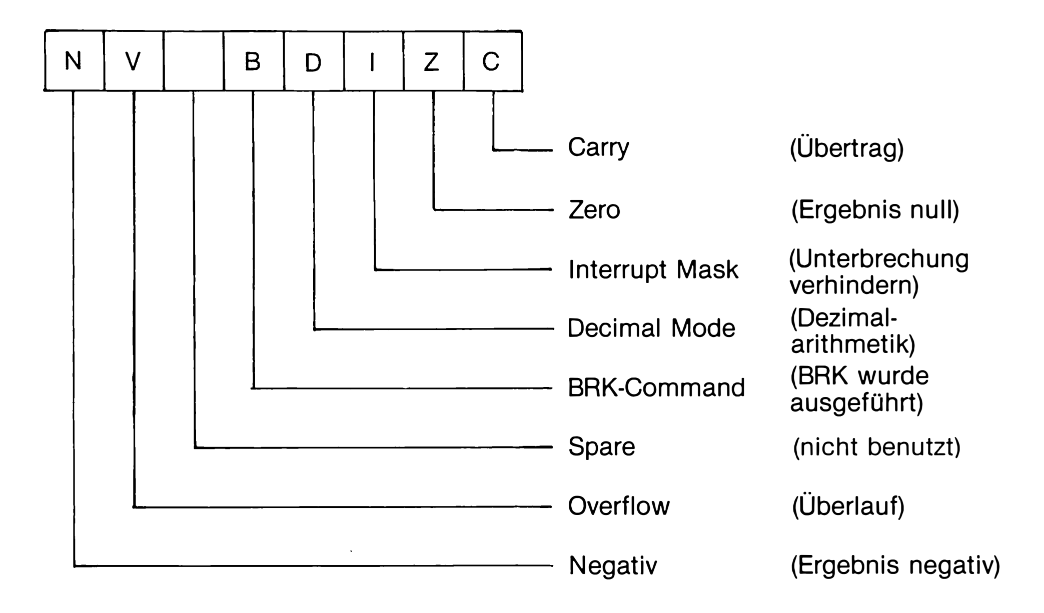
Interrupt-Disable (Unterbrechung verhindern)

Wenn dieses Bit gesetzt ist, können keine Unterbrechungen, die durch den IRQ-Anschluß angefordert werden, bearbeitet werden. Es ist notwendig, dieses Bit zu setzen, wenn eine Operation nicht gestört werden darf, das ist im allgemeinen das Verändern des IRQ-Vektors.

Befehle, die das Interrupt-Flag verändern: BRK, CLI, PLP, RTI, SEI.

Befehle, die durch das Interrupt-Flag beeinflußt werden: PHP.

Aufbau des Statusregisters



Decimal-Flag (Dezimal-Arithmetik)

Dieses Flag wählt zwischen Binär-Arithmetik und BCD-Arithmetik aus. BCD-Arithmetik bedeutet, daß im Akku zwei dezimale Ziffern - jeweils als 4-Bit Zahlen kodiert - stehen. Ist dann bei einer Addition das Ergebnis der hinteren Ziffer größer als neun, so erfolgt ein Übertrag auf die vordere Ziffer, und entsprechend wird das Carry-Flag gesetzt, wenn die Summe der vorderen beiden Ziffern plus dem Übertrag der niedrigeren Ziffern größer als neun ist. Die Subtraktion wird analog gehandhabt.

Befehle, die das Decimal-Flag beeinflussen: CLD, PLP, RTI, SED.

Befehle, die durch das Decimal-Flag beeinflußt werden: ADC, PHP, SBC.

Break-Flag (BRK-Befehlsmerker)

Dieses Flag wird genau dann gesetzt, wenn der BRK-Bfehl ausgeführt wird. Dadurch kann man in der Interrupt/Break-Routine unterscheiden, ob diese Routine durch einen BRK-Befehl oder einen IRQ-Interrupt ausgelöst worden ist.

Befehle, die das Break-Flag beeinflussen: BRK, PLP, RTI.

Befehle, die durch das BRK-Flag beinflußt werden: PHP

Overflow-Flag (Überlauf)

Dieses Bit ist im Zusammenhang mit den arithmetischen Befehlen Addieren und Subtrahieren interessant. Es wird immer dann gesetzt, wenn bei vorzeichenbehafteter 8-Bit-Addition der zulässige Bereich überschritten wird. Bei vorzeichenbehafteter Zahldarstellung sind in acht Bit nur Zahlen im Bereich -128 bis +127 darstellbar. Z.B. eine Addition von +100 mit +100 würde zum Setzen des Overflow-Flags führen.

Befehle, die das Overflow-Flag beeinflussen: ADC, BIT, CLV, PLP, RTI, SBC.

Befehle, die durch das Overflow-Flag beeinflußt werden: BVC, BVS, PHP.

Negative-Flag (Ergebnis negativ)

Der Name dieses Flags ist sinnvoll bei Verwendung von vorzeichenbehafteter Arithmetik. Es ist genau dann gesetzt, wenn das Ergebnis einer Operation negativ ist. Im allgemeinen ist es der Fall, wenn das höchstwertige Bit des veränderten Registers gesetzt ist.

Befehle, die das Negative-Flag beeiflussen: ADC, AND, ASL, BIT, CMP, CPX, CPY, DEC, DEX, DEY, EOR, INC, INX, INY, LDA, LDX, LDY, LSR, ORA, PLA, PLP, ROL, ROR, RTI, SPC, TAX, TAY, TYA, TSX, TXA.

Befehle, die durch das Negative-Flag beeinflußt werden: BMI, BPL, PHP.

1.4 Adressierungsarten

Im Folgenden wollen wir die verschiedenen Adressierungsarten des 6510 genauer erläutern.

Implizite Adressierung

Dies ist die einfachste aller Adressierungsarten. Es wird nämlich bereits durch den Befehl selbst bestimmt, welche Register oder Speicherzellen verändert werden, d.h. es wird keine Adresse benötigt. Befehle mit implizierter Adressierung kennen keine andere Adressierungsart.

Akkumulator

Akkumulator-Adressierung bedeutet, daß der Akkumulator die anzusprechende Adresse ist, und nicht, daß die Adresse im Akkumulator steht. Diese Art gleicht im wesentlichen der impliziten Adressierung, jedoch sind bei den entsprechenden Befehlen noch andere Adressierungsarten möglich. Die Akkumulator-Adressierung wird nur bei Schiebe- und Rotationsbefehlen verwendet.

Unmittelbare (Immediate) Adressierung

Hier folgt unmittelbar auf den Befehlscode der zu ladende Wert, d.h. es wird keine Adresse verwendet. Wenn z.B. das X-Register mit der Konstanten '5' geladen werden soll (LDX #5), so ist dies eine unmittelbare Adressierung.

Absolute Adressierung

Hier folgen dem Befehlscode zwei Bytes, zunächst das Low-Byte, dann das High-Byte, die zusammen die Adresse der zu manipulierenden Speicherzelle angeben. Man benötigt diese Adressierung, wenn man z.B. den Akku mit dem Inhalt einer bestimmten Speicherzelle laden möchte.

Absolut X-indizierte Adressierung

Auch hier folgen dem Befehlscode zwei Bytes, jedoch ist die Adresse des zu manipulierenden Speicherplatzes gegeben durch den Wert der zwei Bytes vermehrt um den Wert des X-Registers. Dadurch kann man leicht Tabellen verwalten, indem die Basisadresse konstant gehalten wird und lediglich das X-Register verändert wird.

Absolute Y-indizierte Adressierung

Diese Adressierungsart gleicht der absolut X-indizierten Adressierung, jedoch wird hier anstatt des X-Registers das Y-Register verwendet.

Zero-Page Adressierung

Hier folgt dem Befehlscode nur ein Byte. Damit wird eine Adresse in den ersten 256 Byte des Speichers (sog. Zero-Page) angesprochen. Dadurch ist kein zweites Byte notwendig und es wird Speicherplatz gespart.

Zero-Page X-indiziert, Zero-Page Y-indiziert

Diese beiden Adresssierungsarten entsprechen im wesentlichen den absolut indizierten Möglichkeiten, jedoch folgt hier ebenfalls nur ein Byte dem Befehlscode, womit dann nur Adressen in der Zero-Page angesprochen werden können.

Relative Adressierung

Diese Adressierung wird bei den bedingten Sprungbefehlen verwendet, indem das Sprungziel durch den momentanen Programmzähler plus oder minus einer Distanz (englisch: Offset) gegeben ist. Werte von 1 bis 127 ergeben Vorwärtssprünge um entsprechend viele Bytes, und Werte von 128 bis 255 entsprechen Rückwärtssprüngen, wobei man hier das Sprungziel wie folgt berechnen muß:

Sprungziel = Programmzähler + Wert des Operanden - 256.

Indirekte Adressierung

Diese Adressierung wird im 6510 nur für Sprungbefehle verwendet. Dabei stellen die auf den Befehlscode folgenden zwei Bytes eine Adresse dar, in der das Sprungziel steht.

Indiziert-indirekte Adressierung (nur mit X-Register)

Dies wird auch als vorindizierte Adressierung bezeichnet und bedeutet, daß das auf den Befehlscode folgende Byte vermehrt um den Wert des X-Registers eine Zelle in den ersten 256 Bytes des Speichers angibt, welche zusammen mit der folgenden Adresse (innerhalb der Zero-Page) die Adresse des gewünschten Operanden angibt.

Indirekt-indizierte Adressierung (nur mit Y-Register)

Diese Möglichkeit mit Indizierung nach der indirekten Adressierung (Nachindizierung) bedeutet, daß das auf den Befehlscode folgende Byte ein Zellenpaar in den ersten 256 Byte des Speichers adressiert, in denen eine Adresse (2 Bytes (Low/High) steht. Die effektive Adresse des Operanden erhält man durch Hinzufügen des Wertes des Y-Registers zu dieser Adresse.

1.5 Die Befehle in alphabetischer Reihenfolge

Dieses Kapitel widmet sich dem Befehlsvorrat des 6510, wobei jeder Befehl mit seiner mnemotechnischen Bezeichnung - aufgrund derer auch sortiert ist - und einer Kurzcharakteristik vorgestellt wird. In der Kurzcharakteristik wer-

den beschrieben:

- die hexadezimalen Operationscodes
- die ursprüngliche englische Bezeichnung
- deutsche Übersetzung
- Die Anzahl der Bytes je Befehl
- die beeinflussten Flags
- die beeinflussenden Flags
- ggf. die Funktion
- Aufschlüsselung nach Adressierungsart

Folgende **Abkürzungen** wurden verwendet (die Sie teilweise schon kennengelernt haben):

*	Flag wird verändert bzw. Multiplizierungsoperator	
&	Logische UND-Verknüpfung	
v	Logische ODER-Verknüpfung	
0	Flag wird gelöscht	
1	Flag wird gesetzt	
B	Break-Flag	BRK-Befehls-Flag
C	Carry-Flag	Übertrag
D	Dezimal-Flag	Dezimal-Arithmetik
I	Interrupt-Flag	Unterbrechungsmaske
M	Memory	Daten
(M)	Inhalt der Speicherzelle M	
M6	Memory-Bit 6	Datenbit Nr. 6
M7	Memory-Bit 7	Datenbit Nr. 7
N	Negativ-Flag	Ergebnis negativ
Op	Operand	
P	Prozessorstatus	Statusregister
PC	Program Counter	Programmzähler
PCL	PC-Low-Byte	Niederwertiges Byte
PCH	PC-High-Byte	Höherwertiges Byte
S	Stackpointer	Stapelzeiger
V	Overflow-Flag	Überlauf
X	X-Register	
Y	Y-Register	
Z	Zero-Flag	Ergebnis null

ADC

Add with Carry — Mit Übertrag addieren

Funktion: A = A+M+C

Der Speicherinhalt und das Übertragsbit werden zum Akkumulator hinzuaddiert. Das Übertragsbit wird dann gesetzt, wenn das Ergebnis den 8-Bit-Bereich überschreitet. Das Overflow-Flag wird gesetzt, wenn der Bereich für eine vorzeichenbehaftete Zahl überschritten wird.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*	*					*	*
Beeinflussende Flags:					*			*

Adressierungsart	Symbol-Form	Code	Bytes
Unmittelbar	ADC #Op	69	2
Zero Page	ADC Op	65	2
Zero Page , X	ADC Op,X	75	2
Absolut	ADC Op	6D	3
Absolut , X	ADC Op,X	7D	3
Absolut , Y	ADC Op,Y	79	3
Vor- indiziert	ADC (Op,X)	61	2
Nach-indiziert	ADC (Op),Y	71	2

AND

And Accu with Memory — Und-Verknüpfung Speicher-Akku

Funktion: A = A & M

Der Akkumulator wird mit den Daten im Speicher logisch Und-verknüpft, d.h. ein bestimmtes Bit wird genau dann gesetzt, wenn das entsprechende Bit im Akkumulator **und** in den Daten des Speichers gesetzt war.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*						*	
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Unmittelbar	AND #Op	29	2
Zero Page	AND Op	25	2
Zero Page , X	AND Op,X	35	2
Absolut	AND Op	2D	3
Absolut , X	AND Op,X	3D	3
Absolut , Y	AND Op,Y	39	3
Vor- indiziert	AND (Op,X)	21	2
Nach-indiziert	AND (Op),Y	31	2

ASL

Arithmetic Shift Left — Ein Bit nach links schieben

Funktion: A = A * 2 bzw. M = M * 2

Die Bits im Akkumulator bzw. der Speicherzelle werden um eine Stelle nach links geschoben. Das niederwertigste Bit wird mit einer Null aufgefüllt, und das höchstwertigste Bit steht anschließend im Carry-Flag. Dies entspricht einer binären Multiplikation mit zwei.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*						*	*
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Accumulator	ASL A	0A	1
Zero Page	ASL Op	06	2
Zero Page , X	ASL Op,X	16	2
Absolut	ASL Op	0E	3
Absolut , X	ASL Op,X	1E	3

BCC

Branch if Carry Clear — Verzweige, wenn Carry = 0

Der Befehl wird dann ausgeführt, wenn das Carry-Flag nicht gesetzt ist. In diesem Fall wird zum aktuellen Programmzähler der Wert des Operanden hinzuaddiert, wobei dieser als vorzeichenbehaftete 8-Bit-Zahl aufzufassen ist, d.h. das höchstwertige Bit hat die Wertigkeit -128.

Der BCC-Befehl wird z.B. dann eingesetzt, wenn bei einem Vergleich der Akkumulator kleiner als der Vergleichswert ist.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:								
Beeinflussende Flags:								*

Adressierungsart	Symbol-Form	Code	Bytes
Relativ	BCC Op	90	2

BCS

Branch if Carry Set — Verzweige, wenn Carry = 1

Das Programm springt um die angegebene Distanz, wenn das Carry-Flag gesetzt ist.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:								
Beeinflussende Flags:								*

Adressierungsart	Symbol-Form	Code	Bytes
Relativ	BCS Op	BO	2

BEQ

Branch if Equal — Verzweige, wenn Zero = 1

Der Sprung wird ausgeführt, wenn das Zero-Flag gesetzt ist. Dieser Befehl wird meist nach Vergleichsoperationen eingesetzt, um die Gleichheit abzuprüfen.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:								
Beeinflussende Flags:							*	

Adressierungsart	Symbol-Form	Code	Bytes
Relativ	BEQ Op	FO	2

BIT

Bit Test — Bits im Speicher mit Akku prüfen

Bit 6 und Bit 7 der angegebenen Speicherzelle werden in das Negativ-Flag und das Overflow-Flag des Status-Registers übertragen. Dann wird der Akkumulator mit der Speicherzelle UND-verknüpft und das Zero-Flag genau dann gesetzt, wenn dieses Ergebnis Null ist, d.h. wenn alle im Akkumulator gesetzten Bits im Speicher nicht gesetzt sind.

Durch diesen Befehl werden weder der Akkumulator noch das X- oder Y-Register verändert. Er dient lediglich zum Setzen der entsprechenden Flags.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	M7	M6					*	
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Zero Page	BIT Op	24	2
Absolut	BIT Op	2C	3

BMI

Branch if Minus — Verzweige, wenn Ergebnis negativ

Das Programm verzweigt, wenn das Negativ-Flag gesetzt ist.

Dieser Befehl wird meist angewendet, wenn nach einem Ladebefehl oder Transferbefehl geprüft werden soll, ob das höchstwertigste Bit (MSB-Most-Significant-Bit) gesetzt ist.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:								
Beeinflussende Flags:	*							

Adressierungsart	Symbol-Form	Code	Bytes
Relativ	BMI Op	30	2

BNE

Branch if Not Equal — Verzweige, wenn Z = 0

Das Programm springt um die angegebene Distanz, wenn das Zero-Flag nicht gesetzt ist. Dieser Fall tritt z.B. ein, wenn bei einem Vergleich 'Nichtübereinstimmung' festgestellt wurde.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:								
Beeinflussende Flags:							*	

Adressierungsart	Symbol-Form	Code	Bytes
Relativ	BNE Op	DO	2

BPL

Branch if Plus — Verzweige, wenn Ergebnis positiv

Dieser Befehl ist das Gegenstück zum Befehl BMI.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:								
Beeinflussende Flags:	*							

Adressierungsart	Symbol-Form	Code	Bytes
Relativ	BPL Op	10	2

BRK

Break — Softwaregesteuerte Unterbrechung

```
Funktion: PC+2 auf Stapel;
          B=1;
          Status auf Stapel;
          I=1;
          PCL = ($FFFE);
          PCH = ($FFFF)
```

Durch diesen Befehl werden zunächst der höherwertige, dann der niederwertige Teil des um zwei erhöhten Programmzäh-

lers auf den Stapel gebracht, anschließend das Break-Flag gesetzt und dann der Status ebenfalls in den Kellerspeicher gebracht. Schließlich wird noch das Interrupt-Disable-Bit gesetzt, um weitere Unterbrechungen zu verhindern. Das Programm springt dann zu der in den Zellen $FFFE und $FFFF angegebenen Adresse. Dieser Befehl arbeitet also ähnlich wie ein hardwareseitig ausgelöster IRQ-Interrupt. Der BRK-Befehl kann jedoch nicht verhindert werden, indem das I-Flag gesetzt wird.

Dieser Befehl wird meist verwendet, um in der Testphase sogenannte Break-Points zu setzen, bei denen das Programm in eine definierte Routine, meistens eine Monitor-Routine springt. Diese Monitor-Routine endet mit RTI, wodurch Status und Programmzähler wieder hergestellt werden.

Es ist zu beachten, daß das Programm dann beim zweiten Byte nach dem BRK-Befehl aufsetzt, da ja PC+2 auf den Stapel gebracht wurde. Deshalb muß das Monitorprogramm den gestapelten Wert entsprechend korrigieren.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:			1			1		
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	BRK	00	1

BVC

Branch if Overflow Clear — Verzweige, wenn V = 0

Das Programm verzweigt, wenn das Overflow-Flag nicht gesetzt ist. Diese Abfrage wird meistens nach einem BIT-Befehl verwendet, oder bei Verwendung von Addition und Subtraktion mit vorzeichenbehafteten 8-Bit-Zahlen.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:								
Beeinflussende Flags:		*						

Adressierungsart	Symbol-Form	Code	Bytes
Relativ	BVC Op	50	2

BVS

Branch if Overflow Set — Verzweige, wenn V = 1

Der Befehl BVS ist die Umkehrung zum Befehl BVC.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:								
Beeinflussende Flags:		*						

Adressierungsart	Symbol-Form	Code	Bytes
Relativ	BVS Op	70	2

CLC

Clear Carry-Flag — Lösche Übertragsbit

Durch diesen Befehl wird das Carry-Flag gelöscht. Er ist immer vor einer Addition anzuwenden, um zu verhindern, daß ein eventuell gesetztes Carry-Bit dazuaddiert wird.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:								0
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	CLC	18	1

CLD

Clear Decimal-Flag Lösche Dezimalarithmetik-Bit

Durch diesen Befehl wird die BCD-Arithmetik abgeschaltet und die normale Binärarithmetik wieder eingeschaltet. Die BCD-Arithmetik wird beim SED-Befehl erklärt.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:					0			
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	CLD	D8	1

CLI

Clear Interrupt Disable Flag Ermögliche Unterbrechungen

Das I-Flag wird gelöscht, wodurch Unterbrechungen angenommen werden können, die durch den Anschluß-Pin IRQ angemeldet werden. Dieser Befehl muß in einer Unterbrechungs-Routine gegeben werden, wenn verschachtelte Unterbrechungen zugelassen werden sollen.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:						0		
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	CLI	58	1

CLV

Clear Overflow Flag — Lösche Übertragsbit

Dieser Befehl löscht das Overflow-Flag. Er ist beim 6510 jedoch relativ sinnlos, höchstens in der Kombination CLV-BVC, wodurch man einen unbedingten Sprung erhält. Der Befehl stammt noch aus dem Befehlssatz des 6502, bei dem das Setzen des Overflow-Flags durch einen Hardware-Anschluß möglich war.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:		0						
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	CLV	B8	1

CMP

Compare to Accumulator — Vergleiche mit Akku

Bei diesem Befehl wird zunächst der Vergleichswert vom Akkumulator abgezogen, das Ergebnis jedoch nicht festgehalten. Der Akkumulator wird also nicht verändert. Jedoch werden die Flags so gesetzt, als ob eine Subtraktion stattgefunden hätte. Das Zero-Flag ist also dann gesetzt, wenn der Akkumulator identisch mit dem Vergleichswert war. Das Carry-Flag wird gesetzt, wenn der Vergleichswert kleiner oder gleich dem Akkumulator war. Das Overflow-Flag wird jedoch nicht verändert. Die Behandlung des Negative-Flags ist meist sinnlos, da das virtuelle Ergebnis der

Subtraktion an sich meist keinen Informationswert besitzt.

Tabellarisch kann man die Verzweigungsmöglichkeiten wie folgt angeben:

```
A kleiner als           M: BCC
A kleiner oder gleich   M: BEQ / BCC
A gleich                M: BEQ
A größer oder gleich    M: BCS
A größer als            M: BEQ $+2, BCS
```

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*						*	*
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Unmittelbar	CMP #Op	C9	2
Zero Page	CMP Op	C5	2
Zero Page , X	CMP Op,X	D5	2
Absolut	CMP Op	CD	3
Absolut , X	CMP Op,X	DD	3
Absolut , Y	CMP Op,Y	D9	3
Vor- indiziert	CMP (Op,X)	C1	2
Nach-indiziert	CMP (Op),Y	D1	2

CPX

Compare to X — Vergleiche mit X-Register

Hier geschieht das gleiche wie beim CMP-Befehl, jedoch wird anstatt dem Akku das X-Register verglichen.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*						*	*
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Unmittelbar	CPX #Op	E0	2
Zero Page	CPX Op	E4	2
Absolut	CPX Op	EC	3

CPY

Compare to Y — Vergleiche mit Y-Register

Dieser Befehl ist ebenfalls analog zum CMP-Befehl, jedoch wird hier das Y-Register anstatt dem Akku verwendet.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*						*	*
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Unmittelbar	CPY #Op	C0	2
Zero Page	CPY Op	C4	2
Absolut	CPY Op	CC	3

DEC

Decrement Memory — Vermindere Speicherzelle um 1

Funktion: M = M - 1

Mit dem DEC-Befehl kann man den Wert einer Speicherzelle um 1 vermindern. Entsprechend dem neuen Wert der Speicherzelle werden Negativ- und Zero-Flag gesetzt.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*						*	
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Zero Page	DEC Op	C6	2
Zero Page , X	DEC Op,X	D6	2
Absolut	DEC Op	CE	3
Absolut , X	DEC Op,X	DE	3

DEX

Decrement X — Vermindere X-Register um 1

Funktion: X = X - 1

Mit diesem Befehl wird das X-Register um eins vermindert. Da das X-Register meist als Schleifenvariable verwendet wird, kann man den Sprung zu einem weiteren Schleifendurchlauf mit den Befehlen BNE oder BPL durchführen, je nach dem, ob die Schleife bis 1 oder bis 0 laufen soll.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*						*	
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	DEX	CA	1

DEY

Decrement Y — Vermindere Y-Register um 1

Funktion: Y = Y - 1

Das Y-Register wird um eins vermindert.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*						*	
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	DEY	88	1

EOR

Exclusive-Or Memory with Accumulator
Akku mit Speicher Exklusiv-Odern

Der Akkumulator wird mit dem Speicher exklusiv-oder-verknüpft, d.h. ein Bit wird im Akku genau dann gesetzt, wenn das entsprechende Bit **entweder** im Akku **oder** im Speicher gesetzt war. Ein Befehl EOR #$FF bewirkt also das Umkehren aller Bits im Akkumulator.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*						*	*
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Unmittelbar	EOR #Op	49	2
Zero Page	EOR Op	45	2
Zero Page , X	EOR Op,X	55	2
Absolut	EOR Op	4D	3
Absolut , X	EOR Op,X	5D	3
Absolut , Y	EOR Op,Y	59	3
Vor- indiziert	EOR (Op,X)	41	2
Nach-indiziert	EOR (Op),Y	51	2

INC

Increment Memory
Erhöhe Speicherzelle um 1

Funktion: M = M + 1

Der Inhalt der angegebenen Speicherzellen wird um eins erhöht und die Flags N und Z entsprechend dem neuen Inhalt gesetzt.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*						*	
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Zero Page	INC Op	E6	2
Zero Page , X	INC Op,X	F6	2
Absolut	INC Op	EE	3
Absolut , X	INC Op,X	FE	3

INX

Increment X — Erhöhe X-Register um 1

Funktion: X = X + 1

Das X-Register wird um eins erhöht. Wenn dieser Befehl nur in einer Schleife verwendet wird, so sollte man das Ende der Schleife mit einem CPX #-Befehl abprüfen.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*						*	
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	INX	E8	1

INY

Increment Y — Eröhe Y-Register um 1

Funktion: Y = Y + 1

Das Y-Register wird um eins erhöht.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*						*	
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	INY	C8	1

JMP

Jump to new location — Unbedingter Sprung

Funktion: PCL = (PC+1);
PCH = (PC+2)

Der Programmzähler wird neu geladen, wodurch ein unbedingter Sprung ausgeführt wird. Es werden dabei keine Flags verändert. Die Adressierung ist auch indirekt möglich, was bedeutet, daß der Operand ein Zellenpaar angibt, in dem die effektive Sprungadresse steht.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:								
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Absolute	JMP Op	4C	3
Indirect	JMP (Op)	6C	3

JSR

Jump to new location saving return adress
Unterprogrammaufruf

Funktion: PC+2 auf Stapel;
PCL = (PC+1);
PCH = (PC+2)

Dieser Befehl entspricht dem Basic-Befehl GOSUB. Er rettet den momentanen Programmzähler und lädt ihn neu mit dem angegebenen Wert. Ein durch den JSR-Befehl aufgerufenes Unterprogramm muß mit RTS beendet werden.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:								
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Absolute	JSR Op	20	3

LDA

Load Accu with Memory   Akku mit Speicherinhalt laden

Funktion: A = M

Der Akkumulator wird mit dem Inhalt der angegebenen Speicherzelle geladen. Das Negativ- und das Zero-Flag werden entsprechend dem Wert verändert.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*						*	
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Unmittelbar	LDA #Op	A9	2
Zero Page	LDA Op	A5	2
Zero Page , X	LDA Op,X	B5	2
Absolut	LDA Op	AD	3
Absolut , X	LDA Op,X	BD	3
Absolut , Y	LDA Op,Y	B9	3
Vor- indiziert	LDA (Op,X)	A1	2
Nach-indiziert	LDA (Op),Y	B1	2

LDX

Load X with Memory — X-Register mit Speicherinhalt laden

Funktion: X = M

Hier wird das X-Register einer Speicherzelle geladen und die Flags N und Z entsprechend verändert.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*						*	
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Unmittelbar	LDX #Op	A2	2
Zero Page	LDX Op	A6	2
Zero Page , Y	LDX Op,Y	B6	2
Absolut	LDX Op	AE	3
Absolut , Y	LDX Op,Y	BE	3

LDY

Load Y with Memory — Y-Register mit Speicherinhalt laden

Funktion: Y = M

Das Y-Register wird mit der angegebenen Speicherzelle geladen und das Negativ- und das Zero-Flag verändert.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*						*	
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Unmittelbar	LDY #Op	A0	2
Zero Page	LDY Op	A4	2
Zero Page , X	LDY Op,X	B4	2
Absolut	LDY Op	AC	3
Absolut , X	LDY Op,X	BC	3

LSR

Logical Shift Right
Akkuinhalt um 1 Bit nach rechts schieben

Funktion: M = M / 2

Der Akkumulator oder eine Speicherzelle werden um ein Bit nach rechts geschoben, d.h. das Bit 0 wird im Carry-Flag abgelegt, und von links wird eine Null in das Bit 7 nachgezogen. Die übrigen Bits wandern jeweils eine Stelle nach rechts. Interpretiert man diese Operation arithmetisch, so entspricht sie einer Division durch 2, wobei der Rest im Carry-Bit steht.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	0						*	*
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Accumulator	LSR A	4A	2
Zero Page	LSR Op	46	2
Zero Page , X	LSR Op,X	56	2
Absolut	LSR Op	4E	3
Absolut , X	LSR Op,X	5E	3

NOP

No Operation   Leerbefehl

Dieser Befehl ist ein Leerbefehl, und benötigt zwei Zyklen, d.h. bei einer Zyklusfrequenz von 1 MHz genau 2 µs. Dieser Befehl wird eingesetzt um Warteschleifen zu bilden oder um in der Testphase Platz zu schaffen für Befehle, die man später einbauen will.

```
                           N  V  B  -  D  I  Z  C
Beeinflusste   Flags:
Beeinflussende Flags:
```

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	NOP	EA	1

ORA

Or Accu with Memory   Akku mit Speicherinhalt 'oderieren'

Funktion: A = A v M

Der Akkumulator wird mit der angegebenen Speicherzelle Oder-verknüpft. Entsprechend dem Ergebenis werden das N- und Z-Flag gesetzt.

```
                           N  V  B  -  D  I  Z  C
Beeinflusste   Flags:      *                 *
Beeinflussende Flags:
```

Adressierungsart	Symbol-Form	Code	Bytes
Unmittelbar	ORA #Op	09	2
Zero Page	ORA Op	05	2
Zero Page , X	ORA Op,X	15	2
Absolut	ORA Op	0D	3
Absolut , X	ORA Op,X	1D	3
Absolut , Y	ORA Op,Y	19	3
Vor- indiziert	ORA (Op,X)	01	2
Nach-indiziert	ORA (Op),Y	11	2

PHA

Push Accumulator on Stack — Akku auf Stapel bringen

Der Akkumulator wird auf den Stapel gebracht. Der Stapelzeiger (S) ist anschließend um eins vermindert.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:								
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	PHA	48	1

PHP

Push Processorstatus on Stack — Bringe Status auf den Stapel

Das Prozessorstatus-Register wird auf den Stapel gebracht. Damit sind alle Flags für eine spätere Benützung zwischengespeichert.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:								
Beeinflussende Flags:	*	*	*	*	*	*	*	*

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	PHP	08	1

PLA

Pull Accumulator from Stack — Hole Akku vom Stapel

Der Akkumulator wird vom Stapel gezogen. Das Negativ- und das Zero-Flag werden entsprechend dem neuen Wert des Akkumulators verändert. Der Stapelzeiger ist anschließend um eins erhöht.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*						*	
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	PLA	68	1

PLP

Pull Processorstatus from Stack — Hole Status vom Stapel

Der Prozessorstatus wird vom Stapel gezogen. Danach können alle Flags verändert sein. Der PLP-Befehl ist das Gegenstück zum PHP-Befehl.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*	*	*	*	*	*	*	*
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	PLP	28	1

ROL

Rotate Left One Bit
Akku bzw. eine Speicherzelle 1 Bit linksherum rotieren

Der Akkumulator bzw. eine Speicherzelle wird zusammen mit dem Carry-Flag um ein Bit nach links rotiert. Das Carry-Flag wird in das Bit 0 übertragen, Bit 0 in Bit 1 usw. bis schließlich Bit 7 in das Carry-Flag übertragen wird. Durch aufeinanderfolgende Rotate-Befehle, die auf verschiedene Speicherzellen wirken, kann man ein Shift-Left von mehr als 8 Bit erreichen.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*						*	*
Beeinflussende Flags:								*

Adressierungsart	Symbol-Form	Code	Bytes
Accumulator	ROL A	2A	2
Zero Page	ROL Op	26	2
Zero Page , X	ROL Op,X	36	2
Absolut	ROL Op	2E	3
Absolut , X	ROL Op,X	3E	3

ROR

Rotate Right One Bit — Akku 1 Bit rechtsherum rotieren

Hier wird eine Speicherzelle oder der Akkumulator um ein Bit nach rechts rotiert. Das Carry-Flag wird in das Bit 7 übertragen, Bit 7 in Bit 6 usw., schließlich wird das Bit 0 in das Carry-Flag übertragen.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*						*	*
Beeinflussende Flags:								*

Adressierungsart	Symbol-Form	Code	Bytes
Accumulator	ROR A	6A	2
Zero Page	ROR Op	66	2
Zero Page , X	ROR Op,X	76	2
Absolut	ROR Op	6E	3
Absolut , X	ROR Op,X	7E	3

RTI

Return from Interrupt — Rückkehr von Unterbrechung

Funktion: P vom Stapel holen ;
PC vom Stapel holen

Dieser Befehl sollte am Ende einer Unterbrechungs- oder einer BRK-Routine stehen. Er stellt den alten Wert des Programmzählers und des Statusregisters wieder her. Es werden also drei Byte vom Stapel gezogen.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*	*	*	*	*	*	*	*
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	RTI	40	1

RTS

Return from Subroutine — Rückkehr von Unterprogramm

Funktion: PC vom Stapel holen ;
PC = PC + 1

Dieser Befehl entspricht dem Basic-Befehl RETURN. Er verzweigt zum aufrufenden Programm. Da bei einem JSR-Befehl PC+2 auf den Stapel gelegt wurde, der JSR-Befehl selbst aber drei Byte beansprucht, muß nach dem Holen des PC vom Stapel dieser noch um eins erhöht werden, wodurch das Programm mit dem auf den JSR-Befel folgenden Befehl weitermacht.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:								
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	RTS	60	1

SBC

Subtract Memory from Accu — Speicher vom Akku abziehen

Funktion: A = A - M - 1 + C

Die Daten der angegebenen Speicherzelle werden vom Akku-

mulator abgezogen. Außerdem wird noch eine 1 abgezogen, wenn das Carry-Bit nicht gesetzt war. Das Negative- und Zero-Bit wird entsprechend dem neuen Wert des Akkumulators gesetzt. Das Carry-Bit wird gelöscht, wenn die Zahl, die abgezogen wird größer war als der alte Wert des Akkumulators. Das Overflow-Flag wird gesetzt, wenn der Bereich für eine vorzeichenbehaftete 8-Bit-Zahl überschritten wurde. Der Befehl arbeitet in binärer Arethmetik, wenn D gleich 0 ist, sonst in BCD-Arithmetik (siehe SED-Befehl).

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*	*					*	*
Beeinflussende Flags:					*			*

Adressierungsart	Symbol-Form	Code	Bytes
Unmittelbar	SBC #Op	E9	2
Zero Page	SBC Op	E5	2
Zero Page , X	SBC Op,X	F5	2
Absolut	SBC Op	ED	3
Absolut , X	SBC Op,X	FD	3
Absolut , Y	SBC Op,Y	F9	3
Vor- indiziert	SBC (Op,X)	E1	2
Nach-indiziert	SBC (Op),Y	F1	2

SEC

Set Carry Flag — Übertrags-Bit setzen

Das Carry-Flag wird gesetzt. Dieser Befehl ist immer vor einer Subtraktion anzuwenden, wenn verhindert werden soll, daß ein eventuell gelöschtes Carry-Bit ein falsches Ergebnis hervorruft.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:								1
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	SEC	38	1

SED

Set Decimal Mode — Dezimalarithmetik einschalten

Durch diesen Befehl wird das Rechenwerk für Addition und Subtraktion auf BCD-Arithmetik umgeschaltet. Das bedeutet, daß im Akku nun zwei, jeweils in 4 Bit kodierte, Dezimalziffern stehen.

Bei einer Addition geschieht dann folgendes: die niederwertigen 4 Bit von Akkumulator und Speicher werden zusammen mit dem Übertrags-Bit addiert. Ein Übertrag auf die höherwertigen 4 Bit entsteht dann, wenn das Ergebnis größer als 9 ist. Anschließend werden die höherwertigen vier Bit mit diesem Übertrag addiert und es entsteht ein Übertrag, wenn insgesamt das Ergebnis größer als $99 ist. Die Subtraktion funktioniert analog.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:					1			
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	SED	F8	1

SEI

Set Interrupt Disable Bit — Verhindere Unterbrechungen

Durch diesen Befehl werden weitere Unterbrechungen, die am IRQ-Anschluß angemeldet werden nicht ausgeführt. Es ist machmal notwendig, zu verhindern, daß ein Programm unterbrochen wird, insbesondere, wenn die Sprungadresse

einer Unterbrechungsroutine geändert werden soll.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:						1		
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	SEI	78	1

STA

Store Accumulator in Memory — Akku speichern

Der Akkumulator wird in die angegebene Speicherzelle geschrieben. Dabei werden keine Flags verändert.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:								
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Zero Page	STA Op	85	2
Zero Page , X	STA Op,X	95	2
Absolut	STA Op	8D	3
Absolut , X	STA Op,X	9D	3
Absolut , Y	STA Op,Y	99	3
Vor- indiziert	STA (Op,X)	81	2
Nach-indiziert	STA (Op),Y	91	2

STX

Store X in Memory — X-Register speichern

Das X-Register wird ohne Veränderung von Flags in die an-

gegebene Speicherzelle geschrieben.

N V B - D I Z C

Beeinflusste Flags:
Beeinflussende Flags:

Adressierungsart	Symbol-Form	Code	Bytes
Zero Page	STX Op	86	2
Zero Page , Y	STX Op,Y	96	2
Absolut	STX Op	8E	3

STY

Store Y in Memory Y-Register speichern

Dieser Befehl funktioniert analog dem STX-Befehl.

N V B - D I Z C

Beeinflusste Flags:
Beeinflussende Flags:

Adressierungsart	Symbol-Form	Code	Bytes
Zero Page	STY Op	84	2
Zero Page , X	STY Op,X	94	2
Absolut	STY Op	8C	3

TAX

Transfer Accu to X Übertrage Akku ins X-Register

Der Inhalt des Akkumulators wird in das X-Register kopiert
und das N- und Z-Flag entsprechend gesetzt.

```
                          N  V  B  -  D  I  Z  C
Beeinflusste   Flags:     *                 *
Beeinflussende Flags:
```

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	TAX	AA	1

TAY

Transfer Accu to Y — Übertrage Akku ins Y-Register

Der Akkumulator wird in das Y-Register kopiert und ebenfalls die Flags entsprechend gesetzt.

```
                          N  V  B  -  D  I  Z  C
Beeinflusste   Flags:     *                 *
Beeinflussende Flags:
```

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	TAY	A8	1

TSX

Transfer Stackpointer to X
Übertrage Stapelzeiger ins X-Register

Der momentane Wert des Stapelzeigers wird in das X-Register übertragen. Dabei werden wieder N- und Z-Flag verändert.

```
                          N  V  B  -  D  I  Z  C
Beeinflusste   Flags:     *                 *
Beeinflussende Flags:
```

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	TSX	BA	1

TXA

Transfer X to Accu   Übertrage X-Registerin den Akku

Das X-Register wird in den Akku übertragen und die Flags N und Z entsprechend angepaßt.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:	*						*	
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	TXA	8A	1

TXS

Transfer X to Stackpointer
Übertrage X-Register in den Stapelzeiger

Dies ist der einzige Befehl, mit dem man den Stapelzeiger laden (beschreiben) kann. Und zwar wird hier das X-Register in den Stapelzeiger übertragen. Dabei werden keine Flags verändert.

	N	V	B	-	D	I	Z	C
Beeinflusste Flags:								
Beeinflussende Flags:								

Adressierungsart	Symbol-Form	Code	Bytes
Implizit	TXS	9A	1

TYA

Transfer Y to Accu Übertrage Y-Register in den Akku

Das Y-Register wird in den Akkumulator kopiert und die Flags entsprechend angepaßt.

```
                          N  V  B  -  D  I  Z  C
Beeinflusste   Flags:     *                 *
Beeinflussende Flags:
```

```
!================================================!
! Adressierungsart ! Symbol-Form ! Code ! Bytes !
!------------------------------------------------!
! Implizit         ! TYA         !  98  !   1   !
!================================================!
```

1.6 Einfache Beispiele für Maschinenprogramme

Wir wollen in diesem Kapitel nur zwei ganz einfache Beispiele für Maschinenprogrammierung vorstellen, zum einen eine 16-Bit Addition, zum anderen das Suchen eines Wertes in einer Tabelle. Mit diesen einfachen Beispielen werden aber bereits die wesentlichen Befehle und Adressierungsarten des 6510 deutlich. Kompliziertere Beispiele können Sie in Kapitel 4 sehen, wo Hilfsroutinen für den Assembler vorgestellt sind. Außerdem sind in Band 1 und Band 3 zahlreiche weitere Beispiele bei den Grafikerweiterungen zu finden.

16-Bit-Addition

```
140  C000  0000      SUM1: WOR 0 ; 1.SUMMAND
150  C002  0000      SUM2: WOR 0 ; 2.SUMMAND
160  C004  0000      SUMME:WOR 0 ; ERGEBNIS
170  C006            ;
180  C006            ADDITION:
190  C006  D8        CLD ; BINAERARITHMETIK EIN
200  C007  18        CLC ; UEBERTRAG LOESCHEN
210  C008  AD00C0    LDA SUM1     ;1.SUM.  LOW-BYTE
220  C00B  6D02C0    ADC SUM2     ;+ 2.SUM.LOW-BYTE
230  C00E  8D04C0    STA SUMME    ;= SUMME LOW-BYTE
240  C011  AD01C0    LDA SUM1+1   ;1.SUM. HIGH-BYTE
250  C014  6D03C0    ADC SUM2+1   ;+2.SUM HIGH-BYTE
260  C017  8D05C0    STA SUMME+1  ;=SUMME HIGH-BYTE
270  C01A  B000    R BCS FEHLER   ;WENN SUMME>$FFFF
280  C01C  60        RTS  ;ENDE DER ROUTINE
```

In diesem kleinen Unterprogramm werden zunächst drei Worte (zwei Byte) definiert, SUM1 und SUM2, die die Summanden der 16-Bit Addition darstellen und als Eingabeparameter für das Programm dienen. In einem weiteren Wort SUMME soll das Ergebnis der Addition abgelegt werden. Das Programm beginnt mit dem Label ADDITION. Dort wird zunächst das Dezimal-Flag gelöscht, da wir ja hier eine binäre 16-Bit-Addition ausführen wollen. Außerdem wird das Carry-Flag gelöscht, da sonst eine weitere eins hinzu addiert werden würde. Diese beiden Befehle können entfallen, wenn durch das Aufrufen des Programms sichergestellt ist, daß diese beiden Flags gelöscht sind.

Das einzige für Arithmetik verwendbare Register ist der Akkumulator. Deshalb wird er mit den niederwertigen acht Bit des ersten Summanden geladen, dann die niederwertigen acht Bit des zweiten Summanden hinzuaddiert und das Ergebnis, das wieder im Akku steht, in die niederwertigen acht Bit des Ergebnisses gespeichert.

Nach dieser Addition kann das Carry-Bit gesetzt sein. Diesen Übertrag können wir bei der Addition der höherwertigen acht Bit mit verarbeiten. Dazu wird zunächst der Akkumulator mit dem höherwertigen Byte des ersten Summanden geladen, und dann dazu das höherwertige Byte des zweiten Summanden addiert. Bei dieser Addition wird der Übertrag bereits mitverarbeitet. Dann wird das Ergebnis, welches wieder im Akkumulator steht in das höherwertige Byte des Ergebnisses gespeichert.

Ist nach dieser Addition das Carry-Bit gesetzt, so bedeutet dies, daß der zulässige Bereich für eine vorzeichenlose 16-Bit-Binärzahl überschritten wurde. Dann muß zu einer Fehlermeldungs-Routine verzweigt werden, die wir hier jedoch nicht angegeben haben. Wurde der zulässige Bereich eingehalten, so wird dann das Unterprogramm mit RTS verlassen.

Es ist immer sinnvoll, eine Fehlerbehandlung für die Fälle vorzusehen, für die das Maschinenprogramm nicht ausgelegt ist. Das gleiche Maschinenprogramm kann man übrigens für die Addition von Vorzeichenbehafteten 16-Bit-Binärzahlen verwenden. Dann muß aber die Fehler-Routine aufgerufen werden, wenn das Overflow-Flag gesetzt ist. Die vorletzte Zeile des Programms muß also geändert werden in:

```
BVS FEHLER.
```

Suchen eines Wertes in einer Tabelle

```
ASSEMBLIEREN    VON TAB.SRC
OBJECT-DATEI    IST TAB.OBJ
SYMBOL-TABELLE IST TAB.SYM

ZEILE  ADR.  OBJ     * QUELLTEXT

  100  C100            %ORG $C100
  110  C100            ;
  120  C100            ;***** SUCHEN IN EINER TABELLE *****
  130  C100            ;
  140  C100  00        SUCH: BYT 0 ;ZU SUCHENDER WERT
  150  C101  00        POS:  BYT 0 ;POSITION DES WERTES
  160  C102            ;
  170  C102            TABANF:
  180  C102  05        BYT 5
  190  C103  08        BYT 8
  200  C104  03        BYT 3
  210  C105  09        BYT 9
  220  C106  07        BYT 7
  230  C107  00        BYT 0
  240  C108  01        BYT 1
  250  C109            TABEND:
  260  C109            ;
  270  C109  A207      LDX #TABEND-TABANF    ;TAB.GROESSE
  280  C10B            SCHLEIFE:
  290  C10B  BD02C1    LDA TABANF,X      ;WERT AUS TABELLE
  300  C10E  CD00C1    CMP SUCH        ;MIT SUCHWERT VERGL.
  310  C111  F000    R BEQ SUCHEND  ;WENN GEF, ZU SUCHEND
  320  C113  CA        DEX      ;NAECHSTES (VORIGES) BYTE
  330  C114  10F5      BPL SCHLEIFE          ; BIS X < 0
  340  C116            SUCHEND:
  350  C116  8E01C1    STX POS           ;ERGEBNIS ABLEGEN
  360  C119  60        RTS               ;ENDE DER ROUTINE
  370  C11A            ;
  370  C11A
DATEIENDE ERREICHT.
```

Das Programm soll folgende Aufgaben erfüllen: Es soll die Position eines 8-Bit-Wertes in einer Tabelle von 8-Bit-Werten bestimmen. Wird der Wert nicht gefunden, so soll die Position $FF ausgegeben werden. Eingabeparameter ist eine Speicherzelle SUCH, in der der zu suchende Wert enthalten ist, Ausgabeparameter ist eine Zelle POS, mit der die Position des Zeichens übergeben wird.

Das Programm muß in einer Schleife alle Werte der Tabelle durchsuchen, bis der gewünschte Wert gefunden ist, oder die Tabelle zu Ende ist. Dazu wird zunächst das X-Register mit der Größe der Tabelle geladen. Die Größe der Tabelle berechnet sich einfach aus TABEND-TABANF. Dann wird der Akku mit dem letzten Zeichen der Tabelle geladen, was hier durch die indizierte Addressierung LDA TABANF,X geschieht. Dieser Wert wird mit dem suchenden Zeichen mit CMP SUCH verglichen. Wurde Übereinstimmung gefunden, so wird zum Ende des Unterprogramms verzweigt, wo das X-Register, welches ja die Position enthält, in die Ergebnis-Zelle POS gespeichert wird.

Bei einer Nicht-Übereinstimmung wird das X-Register um eins vermindert, damit die indizierte Anweisung LDA TABANF,X nun auf das nächste (hier das vorhergehende) Byte der Tabelle zeigt. War das X-Register bereits Null, so enthält es nun den Wert $FF. Dann ist das Negativ-Flag gesetzt, und man kann daran erkennen, daß die Tabelle zu Ende ist. In diesem Fall braucht nur noch das X-Register in das Ergebnis-Register POS abgelegt zu werden, und das Unterprogramm wird verlassen.

Es ist besonders zu beachten, daß dieses Unterprogramm nur für Tabellen bis zu 127 Byte Größe geeignet ist. Für größere Tabellen bis 254 Byte kann das Programm fast ähnlich übernommen werden, wenn man den Befehl BPL SCHLEIFE durch die Befehlsfolge CPX #$FF , BNE SCHLEIFE ersetzt. Für noch größere Tabellen kann man die Tabelle nicht mehr mit einem einzigen Index-Register verwalten, sondern man muß die indirekte Adressierung über ein Zellenpaar in der Zero-Page anwenden.

Die Anwendung dieser indirekt-indizierten Adressierung können Sie z.B. in Kapitel 4.9 verfolgen, wo eine Zeichenreihe in einer anderen gesucht wird. Dies ist ein erweitertes Beispiel für das Suchen eines Zeichens in einer Tabelle.

1.7 Die Befehle in numerischer Reihenfolge

Wir wollen in diesem Kapitel eine Tabelle angeben, die sämtliche Befehle des 6510-Prozessors enthält. Diese Tabelle ist nützlich, wenn Sie ein kleines Programm per Hand disassemblieren wollen. Aus dieser Tabelle sind auch die nicht verwendeten Codes ersichtlich. Zu jedem Befehl wird neben dem Code die mnemotechnische Bezeichnung mit der entsprechenden Adressierungsart angegeben.

```
00 BRK
01 ORA (Op,X)
02 illegal
03 illegal
04 illegal
05 ORA Op
06 ASL Op
07 illegal
08 PHP
09 ORA #Op
0A ASL A
0B illegal
0C illegal
0D ORA Op
0E ASL Op
0F illegal
10 BPL Op
11 ORA (Op),Y
12 illegal
13 illegal
14 illegal
15 ORA Op,X
16 ASL Op,X
17 illegal
18 CLC
19 ORA OP,Y
1A illegal
1B illegal
1C illegal
1D ORA Op,X
1E ASL Op,X
1F illegal
20 JSR Op
21 AND (Op,X)
22 illegal
23 illegal
24 BIT Op
25 AND Op
26 ROL Op
27 illegal
28 PLP
29 AND #Op
2A ROL #Op
2B illegal
2C BIT Op
2D AND Op
2E ROL Op
2F illegal
30 BMI Op
31 AND (Op),Y
32 illegal
33 illegal
34 illegal
35 AND Op,X
36 ROL Op,X
37 illegal
38 SEC
39 AND Op,Y
3A illegal
3B illegal
3C illegal
3D AND Op,X
3E ROL Op,X
3F illegal
40 RTI
41 EOR (Op,X)
42 illegal
43 illegal
44 illegal
45 EOR Op
46 LSR Op
47 illegal
48 PHA
49 EOR #Op
4A LSR A
4B illegal
4C JMP Op
4D EOR Op
4E LSR Op
4F illegal
50 BVC Op
51 EOR (Op),Y
52 illegal
53 illegal
54 illegal
55 EOR Op,X
56 LSR Op,X
57 illegal
58 CLI
59 EOR Op,Y
5A illegal
5B illegal
5C illegal
5D EOR Op,X
5E LSR Op,X
5F illegal
60 RTS
61 ADC (Op,X)
62 illegal
63 illegal
64 illegal
65 ADC Op
```

```
66 ROR Op
67 illegal
68 PLA
69 ADC #Op
6A ROR #Op
6B illegal
6C JMP (Op)
6D ADC Op
6E ROR Op
6F illegal
70 BVS Op
71 ADC (Op),Y
72 illegal
73 illegal
74 illegal
75 ADC Op,X
76 ROR Op,X
77 illegal
78 SEI
79 ADC Op,Y
7A illegal
7B illegal
7C illegal
7D ADC Op,X
7E ROR Op,X
7F illegal
80 illegal
81 STA (Op,X)
82 illegal
83 illegal
84 STY Op
85 STA Op
86 STX Op
87 illegal
88 DEY
89 illegal
8A TXA
8B illegal
8C STY Op
8D STA Op
8E STX Op
8F illegal
90 BCC Op
91 STA (Op),Y
92 illegal
93 illegal
94 STY Op,X
95 STA Op,X
96 STX Op,Y
97 illegal
98 TYA
99 STA Op,Y
9A TXS
9B illegal
9C illegal
9D STA Op,X
9E illegal
9F illegal
A0 LDY #Op
A1 LDA (Op,X)
A2 LDX #Op
A3 illegal
A4 LDY Op
A5 LDA Op
A6 LDX Op
A7 illegal
A8 TAY
A9 LDA #Op
AA TAX
AB illegal
AC LDY Op
AD LDA Op
AE LDX Op
AF illegal
B0 BCS Op
B1 LDA (Op),Y
B2 illegal
B3 illegal
B4 illegal
B5 LDA Op,X
B6 LDX Op,Y
B7 illegal
B8 CLV
B9 LDA Op,Y
BA TSX
BB illegal
BC LDY Op,X
BD LDA Op,Y
BE LDX Op,Y
BF illegal
C0 CPY #Op
C1 CMP (Op,X)
C2 illegal
C3 illegal
C4 CPY Op
C5 CMP Op
C6 DEC Op
C7 illegal
C8 INY
C9 CMP #Op
CA DEX
CB illegal
```

```
CC CPY Op
CD CMP Op
CE DEC Op
CF illegal
D0 BNE Op
D1 CMP (Op),Y
D2 illegal
D3 illegal
D4 illegal
D5 CMP Op,X
D6 DEC Op,X
D7 illegal
D8 CLD
D9 CMP Op,X
DA illegal
DB illegal
DC illegal
DD CMP Op,X
DE DEC Op,X
DF illegal
E0 CPX #Op
E1 SBC (Op,X)
E2 illegal
E3 illegal
E4 CPX Op
E5 SBC Op
E6 INC Op
E7 illegal
E8 INX
E9 SBC #Op
EA NOP
EB illegal
EC CPX Op
ED SBC Op
EE INC Op
EF illegal
F0 BEQ Op
F1 SBC (Op),Y
F2 illegal
F3 illegal
F4 illegal
F5 SBC Op,X
F6 INC Op,X
F7 illegal
F8 SED
F9 SBC Op,Y
FA illegal
FB illegal
FC illegal
FD SBC Op,X
FE INC Op,X
FF illegal
```

1.8 Basic-Programm zur Erläuterung

Die meisten Leser werden mit Basic vertraut sein, aber trotz der vorhergehenden Erläuterung noch ein paar Schwierigkeiten mit der Funktion der einzelnen Maschinenbefehle haben. Wir wollen deshalb im folgenden ein Basic-Programm vorstellen, welches die Maschinenbefehle simuliert. Den Prozessorregistern entsprechen dabei einfache Variablen und der Stapel wird mit Hilfe eines Arrays simuliert. Die momentanen Werte der Prozessorregister werden am Bildschirm in hexadezimaler Form ausgegeben. Durch die Verwendung von Basic dauert die Ausführung eines Befehls sehr lang, d.h. etwa 100000 mal länger, als der Prozessor selbst arbeitet. Dies ist aber gerade dazu gut, um in einer Art Einzelschrittverfahren die Funktion eines Programms nachzuvollziehen.

Das Programm wird mit

```
LOAD"DEMO6510",8
```

geladen. Danach fragt das Programm nach der Startadresse,

ab der ein Programm bearbeitet werden soll. Das zu testende Maschinenprogramm muß also bereits im Speicher stehen. Man kann sowohl eingebaute ROM-Routinen testen, als auch mit einen Assembler erstellte Programme nachvollziehen.

Auch ist es möglich, kleine Programme mit POKE-Befehlen einzugeben und dann ab der entsprechenden Speicherzelle den Test zu starten.

Im Anschluß werden die Werte der Prozessorregister, die an das Programm übergeben werden sollen, erfragt. Dabei kann man den Wert des Prozessorstatusregisters in binärer Form eingeben. Schließlich werden noch bis zu 10 Adressen erfragt, bei denen das Programm anhalten soll. Wird im Programmablauf eine dieser Adressen erreicht, so hält das Programm an. Alle Eingaben - außer den Daten des Prozessorstatusregisters - müssen in hexadezimaler Form erfolgen.

Nun wird das Programm schrittweise abgearbeitet und der aktuelle Stand der Prozessorregister nebeneinander auf dem Bildschirm angezeigt. Durch Drücken der SHIFT-Taste kann man das Programm kurzzeitig anhalten.

```
10 PRINT"■   PC   NV-BDIZC   AC  XR  YR  SP
12 HE$="0123456789ABCDEF"
14 DIMST%(255):REM STACK
16 DEFFNEO(M)=(M AND (NOT A)) OR ((NOT M ) AND A)
18 DEFFNV(X)=X-48+7*(X>64)
20 DEFFNF0(X)=(ASC(MID$(H0$,X,1))-48) AND 1
22 GOSUB2000
24 INPUT"■■";H0$
26 HH$=LEFT$(H0$,4):GOSUB400:PC=HH
28 N=FNF0(7)
30 V=FNF0(8)
32 PS=FNF0(9)
34 B=FNF0(10)
36 D=FNF0(11)
38 I=FNF0(12)
40 Z=FNF0(13)
42 C=FNF0(14)
44 H$=MID$(H0$,18,2):GOSUB350:A=H
46 H$=MID$(H0$,22,2):GOSUB350:X=H
48 H$=MID$(H0$,26,2):GOSUB350:Y=H
50 H$=MID$(H0$,30,2):GOSUB350:S=H
52 PRINT"■■"
54 HP=HP+1
56 PRINTHP"■■. ABBRUCHPUNKT";
58 INPUT"     ■■■■";HH$
60 IFHH$="  "THENHP=HP-1:GOTO68
62 GOSUB400
```

```
64 HP(HP)=HH
66 GOTO54
68 PRINT"]
70 GOSUB49000
72 GOTO1000
150 REM *** WERT IN HIGH- UND LOW ZERLEGEN ***
160 HI=INT(HH/256)
170 LO=HH-256*HI
180 RETURN
200 REM *   2-STELLIGE HEXZAHL (H$) AUS H BILDEN          *
210 H$=MID$(HE$,H/16+1,1)+MID$(HE$,(HAND15)+1,1)
220 RETURN
250 REM *   4-STELLIGE HEXZAHL (HH$) AUS HH BILDEN        *
260 H=INT(HH/256)
270 GOSUB200
280 HH$=H$
290 H=HH-256*H
300 GOSUB200
310 HH$=HH$+H$
320 RETURN
350 REM * WERT VON 2-STELLIGER HEX-ZAHL (H$) -> H         *
360 H=16*FNV(ASC(H$))+FNV(ASC(MID$(H$,2)))
370 RETURN
400 REM * WERT VON 4-STELLIGER HEX-ZAHL (HH$) -> HH       *
410 H$=LEFT$(HH$,2)
420 GOSUB350
430 HH=256*H
440 H$=RIGHT$(HH$,2)
450 GOSUB350
460 HH=HH+H
470 RETURN
1000 REM *** SCHLEIFE FUER JEDEN BEFEHL ***
1010 GOSUB2000
1020 IFHP=0THEN1000
1030 FORJ=1TOHP:IFPC=HP(J)THEN1080
1035 NEXTJ
1040 OP=PEEK(PC)
1045 H=OP:GOSUB200:PRINT"OPCODE: "H$"   ";
1050 GOSUB1100
1055 PC=PC+1
1060 GOTO 1000
1080 PRINT"ABBRUCH (";J;")  ";
1090 STOP
```

```
1100 IFOP=0THEN10000
1110 ONOPGOTO10100,1400,1400,1400,10500,10600,1400,10800,10900,11000
1120 ONOP-10GOTO1400,1400,11300,11400,1400,11600,11700,1400,1400,1400
1130 ONOP-20GOTO12100,12200,1400,12400,12500,1400,1400,1400,12900,13000
1140 ONOP-30GOTO1400,13200,13300,1400,1400,13600,13700,13800,1400,14000
1150 ONOP-40GOTO14100,14200,1400,14400,14500,14600,1400,14800,14900,1400
1160 ONOP-50GOTO1400,1400,15300,15400,1400,15600,15700,1400,1400,1400
1170 ONOP-60GOTO16100,16200,1400,16400,16500,1400,1400,1400,16900,17000
1180 ONOP-70GOTO1400,17200,17300,17400,1400,17600,17700,17800,1400,18000
1190 ONOP-80GOTO18100,1400,1400,1400,18500,18600,1400,18800,18900,1400
1200 ONOP-90GOTO1400,1400,19300,19400,1400,19600,19700,1400,1400,1400
1210 ONOP-100GOTO20100,20200,1400,20400,20500,20600,1400,20800,20900,21000
1220 ONOP-110GOTO1400,21200,21300,1400,1400,1400,21700,21800,1400,22000
1230 ONOP-120GOTO22100,1400,1400,1400,22500,22600,1400,1400,22900,1400
1240 ONOP-130GOTO1400,23200,23300,23400,1400,23600,1400,23800,1400,24000
1250 ONOP-140GOTO24100,24200,1400,24400,24500,1400,1400,24800,24900,25000
1260 ONOP-150GOTO1400,25200,25300,25400,1400,1400,25700,1400,1400,26000
1270 ONOP-160GOTO26100,26200,1400,26400,26500,26600,1400,26800,26900,27000
1280 ONOP-170GOTO1400,27200,27300,27400,1400,27600,27700,1400,1400,28000
1290 ONOP-180GOTO28100,28200,1400,28400,28500,28600,1400,28800,28900,29000
1300 ONOP-190GOTO1400,29200,29300,1400,1400,29600,29700,29800,1400,30000
1310 ONOP-200GOTO30100,30200,1400,30400,30500,30600,1400,30800,30900,1400
1320 ONOP-210GOTO1400,1400,31300,31400,1400,31600,31700,1400,1400,1400
1330 ONOP-220GOTO32100,32200,1400,32400,32500,1400,1400,32800,32900,33000
1340 ONOP-230GOTO1400,33200,33300,33400,1400,33600,33700,33800,1400,34000
1350 ONOP-240GOTO34100,1400,1400,1400,34500,34600,1400,34800,34900,1400
1360 ONOP-250GOTO1400,1400,35300,35400,1400
1370 STOP
1400 PRINT"UNGUELTIGER CODE";
1410 STOP
```

```
2000 REM *** ANZEIGE ***
2010 HH=PC:GOSUB250:PC$=HH$
2020 ST$=CHR$(48+N)+CHR$(48+V)+CHR$(48+P5)+CHR$(48+B)
2030 ST$=ST$+CHR$(48+D)+CHR$(48+I)+CHR$(48+Z)+CHR$(48+C)
2040 H=A:GOSUB200:A$=H$
2050 H=X:GOSUB200:X$=H$
2060 H=Y:GOSUB200:Y$=H$
2070 H=S:GOSUB200:S$=H$
2090 PRINT"[illegible]  "PC$"  "ST$"   "A$"  "X$"  "Y$"  "S$
2100 RETURN
3000 REM *** 1-BYTE-OPERAND HOLEN ***
3010 PC=PC+1
3020 O=PEEK(PC)
3030 RETURN
4000 REM *** 2-BYTE-OPERAND HOLEN ***
4010 PC=PC+1
4020 O=PEEK(PC)
4030 PC=PC+1
4040 OO=O+256*PEEK(PC)
4050 RETURN
5000 REM *** STATUS ZUSAMMENFASSEN ***
5010 P=128*N+64*V+32*P5+16*B+8*D+4*I+2*Z+1*C
5020 RETURN
6000 REM *** STATUS ZERLEGEN ***
6010 N=(P AND 128)/128
6020 V=(P AND 64)/64
6030 P5=(P AND 32)/32
6040 B=(P AND 16)/16
6050 D=(P AND 8)/8
6060 I=(P AND 4)/4
6070 Z=(P AND 2)/2
6080 C=P AND 1
6090 RETURN
7000 REM *** RELATIVEN SPRUNG AUSFUEHREN ***
7010 IFO>127THENO=O-256
7020 PC=PC+O
7030 RETURN
8000 REM *** A=A+M+C (ADC-BEFEHL) ***
8010 IF D=0 THEN A=A+M+C : C=-(A>255) : GOTO8070
8020 H=(A AND 15) + (M AND 15) + C
8030 C = -(H>9)
8040 H=H AND 15
8050 A=(A AND 240) + (M AND 240) + H + 16*C
8060 C = -(A>99)
8070 A = A AND 255
8080 N = (A AND 128)/128
8090 Z = -(A=0)
8100 RETURN
8200 REM *** A=A-M-1+C (SBC-BEFEHL) ***
8210 IF D=0 THEN A=A-M-1+C : C=-(A>=0) : GOTO8270
8220 H=(A AND 15) - (M AND 15) - 1 + C
```

```
8230 C = -(H>=0)
8240 H=H AND 15
8250 A=(A AND 240) - (M AND 240) + H - 16 + 16*C
8260 C = -(A>=0)
8270 A = A AND 255
8280 N = (A AND 128)/128
8290 Z = -(A=0)
8300 RETURN
8400 REM *** A-M (CMP-BEFEHL) ***
8410 C = -((A-M)>=0)
8420 N = ((A-M) AND 128)/128
8430 Z = -((A-M)=0)
8440 RETURN
10000 REM *** BRK
10005 HH=PC+2:GOSUB150
10010 ST%(S)=HI:S=(S-1) AND 255
10015 ST%(S)=LO:S=(S-1) AND 255
10020 B=1
10025 GOSUB5000
10030 ST%(S)=P:S=(S-1) AND 255
10035 I=1
10040 PC=PEEK(65534)+256*PEEK(65535)
10045 GOSUB2000
10050 PRINT"BRK ERREICHT";
10055 END
10100 REM *** ORA (INDIRECT,X)
10110 GOSUB3000
10120 OO=PEEK(O+X)+256*PEEK(O+X+1)
10130 A=A OR PEEK(OO)
10140 N=(A AND 128)/128
10150 Z=-(A=0)
10160 RETURN
10500 REM *** ORA ZEROPAGE
10510 GOSUB3000
10520 A=A OR PEEK(O)
10530 N=(A AND 128)/128
10540 Z=-(A=0)
10550 RETURN
10600 REM *** ASL ZEROPAGE
10610 GOSUB3000
10620 M=PEEK(O)
10630 M=M*2
10640 C=-(M>255)
10650 M=M AND 255
10660 POKEO,M
10670 Z=-(M=0)
10680 N=(M AND 128)/128
10690 RETURN
10800 REM *** PHP
10810 GOSUB5000
10820 ST%(S)=P:S=(S-1) AND 255
```

```
10830 RETURN
10900 REM *** ORA #IMMEDIATE
10910 GOSUB3000
10920 A=A OR O
10930 N=(A AND 128)/128
10940 Z=-(A=0)
10950 RETURN
11000 REM *** ASL A
11010 A=A*2
11020 C=-(A>255)
11030 A=A AND 255
11040 Z=-(A=0)
11050 N=(A AND 128)/128
11060 RETURN
11300 REM *** ORA ABSOLUTE
11310 GOSUB4000
11320 A=A OR PEEK(OO)
11330 N=(A AND 128)/128
11340 Z=-(A=0)
11350 RETURN
11400 REM *** ASL ABSOLUTE
11410 GOSUB4000
11420 M=PEEK(OO)
11430 M=M*2
11440 C=-(M>255)
11450 M=M AND 255
11460 POKEOO,M
11470 Z=-(M=0)
11480 N=(M AND 128)/128
11490 RETURN
11600 REM *** BPL
11610 GOSUB3000
11620 IFN=0THENGOSUB7000
11630 RETURN
11700 REM *** ORA (INDIRECT),Y
11710 GOSUB3000
11720 OO=PEEK(O)+256*PEEK(O+1)+Y
11730 A=A OR PEEK(OO)
11740 N=(A AND 128)/128
11750 Z=-(A=0)
11760 RETURN
12100 REM *** ORA ZEROPAGE,X
12110 GOSUB3000
12120 A=A OR PEEK(O+X)
12130 N=(A AND 128)/128
12140 Z=-(A=0)
12150 RETURN
12200 REM *** ASL ZEROPAGE,X
12210 GOSUB3000
12220 M=PEEK(O+X)
12230 M=M*2
```

```
12240 C=-(M>255)
12250 M=M AND 255
12260 POKEO+X,M
12270 Z=-(M=0)
12280 N=(M AND 128)/128
12290 RETURN
12400 REM *** CLC
12410 C=0
12420 RETURN
12500 REM *** ORA ABSOLUTE,Y
12510 GOSUB4000
12520 A=A OR PEEK(OO+Y)
12530 N=(A AND 128)/128
12540 Z=-(A=0)
12550 RETURN
12900 REM *** ORA ABSOLUTE,X
12910 GOSUB4000
12920 A=A OR PEEK(OO+X)
12930 N=(A AND 128)/128
12940 Z=-(A=0)
12950 RETURN
13000 REM *** ASL ABSOLUTE,X
13010 GOSUB4000
13020 M=PEEK(OO+X)
13030 M=M*2
13040 C=-(M>255)
13050 M=M AND 255
13060 POKEOO+X,M AND 255
13070 Z=-(M=0)
13080 N=(M AND 128)/128
13090 RETURN
13200 REM *** JSR
13210 HH=PC+2:GOSUB150
13220 ST%(S)=HI:S=(S-1) AND 255
13230 ST%(S)=LO:S=(S-1) AND 255
13240 GOSUB4000
13250 PC=OO-1
13260 RETURN
13300 REM *** AND (INDIRECT,X)
13310 GOSUB3000
13320 OO=PEEK(O+X)+256*PEEK(O+X+1)
13330 A=A AND PEEK(OO)
13340 N=(A AND 128)/128
13350 Z=-(A=0)
13360 RETURN
13600 REM *** BIT ZEROPAGE
13610 GOSUB3000
13620 M=PEEK(O)
13630 N=(M AND 128)/128
13640 V=(M AND 64)/64
13650 Z=-((A AND M) = 0)
```

```
13660 RETURN
13700 REM *** AND ZEROPAGE
13710 GOSUB3000
13720 A=A AND PEEK(O)
13730 N=(A AND 128)/128
13740 Z=-(A=0)
13750 RETURN
13800 REM *** ROL ZEROPAGE
13810 GOSUB3000
13820 M=PEEK(O)
13830 M=M*2+C
13840 C=-(M>255)
13850 M=M AND 255
13860 POKEO,M AND 255
13870 Z=-(M=0)
13880 N=(M AND 128)/128
13890 RETURN
14000 REM *** PLP
14010 S=(S+1) AND 255
14020 P=ST%(S)
14030 GOSUB6000
14040 RETURN
14100 REM *** AND #IMMEDIATE
14110 GOSUB3000
14120 A=A AND O
14130 N=(A AND 128)/128
14140 Z=-(A=0)
14150 RETURN
14200 REM *** ROL A
14210 A=A*2+C
14220 C=-(A>255)
14230 A=A AND 255
14240 Z=-(A=0)
14250 N=(A AND 128)/128
14260 RETURN
14400 REM *** BIT ABSOLUTE
14410 GOSUB4000
14420 M=PEEK(OO)
14430 N=(M AND 128)/128
14440 V=(M AND 64)/64
14450 Z=-((A AND M) = 0)
14460 RETURN
14500 REM *** AND ABSOLUTE
14510 GOSUB4000
14520 A=A AND PEEK(OO)
14530 N=(A AND 128)/128
14540 Z=-(A=0)
14550 RETURN
14600 REM *** ROL ABSOLUTE
14610 GOSUB4000
14620 M=PEEK(OO)
```

```
14630 M=M*2+C
14640 C=-(M>255)
14650 M=M AND 255
14660 POKEOO,M AND 255
14670 Z=-(M=0)
14680 N=(M AND 128)/128
14690 RETURN
14800 REM *** BMI
14810 GOSUB3000
14820 IFN=1THENGOSUB7000
14830 RETURN
14900 REM *** AND (INDIRECT),Y
14910 GOSUB3000
14920 OO=PEEK(O)+256*PEEK(O+1)+Y
14930 A=A AND PEEK(OO)
14940 N=(A AND 128)/128
14950 Z=-(A=0)
14960 RETURN
15300 REM *** AND ZEROPAGE,X
15310 GOSUB3000
15320 A=A AND PEEK(O+X)
15330 N=(A AND 128)/128
15340 Z=-(A=0)
15350 RETURN
15400 REM *** ROL ZEROPAGE,X
15410 GOSUB3000
15420 M=PEEK(O+X)
15430 M=M*2+C
15440 C=-(M>255)
15450 M=M AND 255
15460 POKEO+X,M AND 255
15470 Z=-(M=0)
15480 N=(M AND 128)/128
15490 RETURN
15600 REM *** SEC
15610 C=1
15620 RETURN
15700 REM *** AND ABSOLUTE,Y
15736 GOSUB4000
15720 A=A AND PEEK(OO+Y)
15730 N=(A AND 128)/128
15740 Z=-(A=0)
15750 RETURN
16100 REM *** AND ABSOLUTE,X
16110 GOSUB4000
16120 A=A AND PEEK(OO+X)
16130 N=(A AND 128)/128
16140 Z=-(A=0)
16150 RETURN
16200 REM *** ROL ABSOLUTE,X
16210 GOSUB4000
```

```
16220 M=PEEK(OO+X)
16230 M=M*2+C
16240 C=-(M>255)
16250 M=M AND 255
16260 POKEOO+X,M AND 255
16270 Z=-(M=0)
16280 N=(M AND 128)/128
16290 RETURN
16400 REM *** RTI
16410 S=(S+1) AND 255
16420 P=ST%(S)
16430 GOSUB6000
16440 S=(S+1) AND 255
16450 PC=ST%(S)
16460 S=(S+1) AND 255
16470 PC=256*PC+ST%(S)
16480 RETURN
16500 REM *** EOR (INDIRECT,X)
16510 GOSUB3000
16520 OO=PEEK(O+X)+256*PEEK(O+X+1)
16530 A=FNEO(PEEK(OO))
16540 N=(A AND 128)/128
16550 Z=-(A=0)
16560 RETURN
16900 REM *** EOR ZEROPAGE
16910 GOSUB3000
16920 A=FNEO(PEEK(O))
16930 N=(A AND 128)/128
16940 Z=-(A=0)
16950 RETURN
17000 REM *** LSR ZEROPAGE
17010 GOSUB3000
17020 M=PEEK(O)
17030 C=M AND 1
17040 M=INT(M/2)
17050 POKEO,M
17060 Z=-(M=0)
17070 N=(M AND 128)/128
17080 RETURN
17200 REM *** PHA
17210 ST%(S)=A:S=(S-1) AND 255
17220 RETURN
17300 REM *** EOR #IMMEDIATE
17310 GOSUB3000
17320 A=FNEO(O)
17330 N=(A AND 128)/128
17340 Z=-(A=0)
17350 RETURN
17400 REM *** LSR A
17410 C=A AND 1
17420 A=INT(A/2)
```

```
17430 Z=-(A=0)
17440 N=(A AND 128)/128
17450 RETURN
17600 REM *** JMP ABSOLUTE
17610 GOSUB4000
17620 PC=OO-1
17630 RETURN
17700 REM *** EOR ABSOLUTE
17710 GOSUB4000
17720 A=FNEO(PEEK(OO))
17730 N=(A AND 128)/128
17740 Z=-(A=0)
17750 RETURN
17800 REM *** LSR ABSOLUTE
17810 GOSUB4000
17820 M=PEEK(OO)
17830 C=M AND 1
17840 M=INT(M/2)
17850 POKEOO,M
17860 Z=-(M=0)
17870 N=(M AND 128)/128
17880 RETURN
18000 REM *** BVC
18010 GOSUB3000
18020 IFV=0THENGOSUB7000
18030 RETURN
18100 REM *** EOR (INDIRECT),Y
18110 GOSUB3000
18120 OO=PEEK(O)+256*PEEK(O+1)+Y
18130 A=FNEO(PEEK(OO))
18140 N=(A AND 128)/128
18150 Z=-(A=0)
18160 RETURN
18500 REM *** EOR ZEROPAGE,X
18510 GOSUB3000
18520 A=FNEO(PEEK(O+X))
18530 N=(A AND 128)/128
18540 Z=-(A=0)
18550 RETURN
18600 REM *** LSR ZEROPAGE,X
18610 GOSUB3000
18620 M=PEEK(O+X)
18630 C=M AND 1
18640 M=INT(M/2)
18650 POKEO+X,M
18660 Z=-(M=0)
18670 N=(M AND 128)/128
18680 RETURN
18800 REM *** CLI
18810 I=0
18820 RETURN
```

```
18900 REM *** EOR ABSOLUTE,Y
18910 GOSUB4000
18920 A=FNEO(PEEK(OO+Y))
18930 N=(A AND 128)/128
18940 Z=-(A=0)
18950 RETURN
19300 REM *** EOR ABSOLUTE,X
19310 GOSUB4000
19320 A=FNEO(PEEK(OO+X))
19330 N=(A AND 128)/128
19340 Z=-(A=0)
19350 RETURN
19400 REM *** LSR ABSOLUTE,X
19410 GOSUB4000
19420 M=PEEK(OO+X)
19430 C=M AND 1
19440 M=INT(M/2)
19450 POKEOO+X,M
19460 Z=-(M=0)
19470 N=(M AND 128)/128
19480 RETURN
19600 REM *** RTS
19610 S=(S+1)AND255 : HI=ST%(S)
19620 S=(S+1)AND255 : LO=ST%(S)
19630 PC=256*HI+LO
19640 REM  PC=PC+1 GESCHIEHT BEI 1040
19650 RETURN
19700 REM *** ADC (INDIRECT,X)
19710 GOSUB3000
19720 OO=PEEK(O+X)+256*PEEK(O+X+1)
19730 M=PEEK(OO)
19740 GOSUB 8000
19750 RETURN
20100 REM *** ADC ZEROPAGE
20110 GOSUB3000
20120 M=PEEK(O)
20130 GOSUB8000
20140 RETURN
20200 REM *** ROR ZEROPAGE
20210 GOSUB3000
20220 M=PEEK(O)
20230 M=(128*C) + M/2
20240 C=-(M<>INT(M))
20250 M=M AND 255
20260 POKEO,M
20270 Z=-(M=0)
20280 N=(M AND 128)/128
20290 RETURN
20400 REM *** PHP
20410 S=(S+1)AND255
20420 A=ST%(S)
```

```
20430 RETURN
20500 REM *** ADC #IMMEDIATE
20510 GOSUB3000
20520 M=O
20530 GOSUB8000
20540 RETURN
20600 REM *** ROR A
20610 A=(128*C) + A/2
20620 C=-(A<>INT(A))
20630 A=A AND 255
20640 Z=-(A=0)
20650 N=(A AND 128)/128
20660 RETURN
20800 REM *** JMP INDIRECT
20810 GOSUB4000
20820 PC=PEEK(OO)+256*PEEK(OO+1)-1
20830 RETURN
20900 REM *** ADC ABSOLUTE
20910 GOSUB4000
20920 M=PEEK(OO)
20930 GOSUB8000
20940 RETURN
21000 REM *** ROR ABSOLUTE
21010 GOSUB4000
21020 M=PEEK(OO)
21030 M=(128*C) + M/2
21040 C=-(M<>INT(M))
21050 M=M AND 255
21060 POKEOO,M
21070 Z=-(M=0)
21080 N=(M AND 128)/128
21090 RETURN
21200 REM *** BVS
21210 GOSUB3000
21220 IFV=1THENGOSUB7000
21230 RETURN
21300 REM *** ADC (INDIRECT),Y
21310 GOSUB3000
21320 OO=PEEK(O)+256*PEEK(O+1)+Y
21330 M=PEEK(OO)
21340 GOSUB8000
21350 RETURN
21700 REM *** ADC ZEROPAGE,X
21710 GOSUB3000
21720 M=PEEK(O+X)
21730 GOSUB8000
21740 RETURN
21800 REM *** ROR ZEROPAGE,X
21810 GOSUB3000
21820 M=PEEK(O+X)
21830 M=(128*C) + M/2
```

```
21840 C=-(M<>INT(M))
21850 M=M AND 255
21860 POKEO+X,M
21870 Z=-(M=0)
21880 N=(M AND 128)/128
21890 RETURN
22000 REM *** SEI
22010 I=1
22020 RETURN
22100 REM *** ADC ABSOLUTE,Y
22110 GOSUB4000
22120 M=PEEK(OO+Y)
22130 GOSUB8000
22140 RETURN
22500 REM *** ADC ABSOLUTE,X
22510 GOSUB4000
22520 M=PEEK(OO+X)
22530 GOSUB8000
22540 RETURN
22600 REM *** ROR ABSOLUTE,X
22610 GOSUB4000
22620 M=PEEK(OO+X)
22630 M=(128*C) + M/2
22640 C=-(M<>INT(M))
22650 M=M AND 255
22660 POKEOO+X,M AND 255
22670 Z=-(M=0)
22680 N=(M AND 128)/128
22690 RETURN
22900 REM *** STA (INDIRECT,X)
22910 GOSUB3000
22920 OO=PEEK(O+X)+256*PEEK(O+X+1)
22930 POKEOO,A
22940 RETURN
23150 RETURN
23200 REM *** STY ZEROPAGE
23210 GOSUB3000
23220 POKEO,Y
23230 RETURN
23300 REM *** STA ZEROPAGE
23310 GOSUB3000
23320 POKEO,A
23330 RETURN
23400 REM *** STX ZEROPAGE
23410 GOSUB3000
23420 POKEO,X
23430 RETURN
23600 REM *** DEY
23610 Y=(Y-1) AND 255
23620 Z=-(Y=0)
23630 N=(Y AND 128)/128
```

```
23640 RETURN
23800 REM *** TXA
23810 A=X
23820 Z=-(A=0)
23830 N=(A AND 128)/128
23840 RETURN '
24000 REM *** STY ABSOLUTE
24010 GOSUB4000
24020 POKEOO,Y
24030 RETURN
24100 REM *** STA ABSOLUTE
24110 GOSUB4000
24120 POKEOO,A
24130 RETURN
24200 REM *** STX ABSOLUTE
24210 GOSUB4000
24220 POKEOO,X
24230 RETURN
24400 REM *** BCC
24410 GOSUB3000
24420 IFC=0THENGOSUB7000
24430 RETURN
24500 REM *** STA (INDIRECT),Y
24510 GOSUB3000
24520 OO=PEEK(O)+256*PEEK(O+1)+Y
24530 POKEOO,A
24540 RETURN
24800 REM *** STY ZEROPAGE,X
24810 GOSUB3000
24820 POKEO+X,Y
24830 RETURN
24900 REM *** STA ZEROPAGE,X
24910 GOSUB3000
24920 POKEO+X,A
24930 RETURN
25000 REM *** STX ZEROPAGE,Y
25010 GOSUB3000
25020 POKEO+Y,X
25030 RETURN
25200 REM *** TYA
25210 A=Y
25220 Z=-(A=0)
25230 N=(A AND 128)/128
25240 RETURN
25300 REM *** STA ABSOLUTE,Y
25310 GOSUB4000
25320 POKEOO+Y,A
25330 RETURN
25400 REM *** TXS
25410 S=X
25420 RETURN
```

```
25700 REM *** STA ABSOLUTE,X
25710 GOSUB4000
25720 POKEOO+X,A
25730 RETURN
26000 REM *** LDY #IMMEDIATE
26010 GOSUB3000
26020 Y=O
26030 N=(Y AND 128)/128
26040 Z=-(Y=0)
26050 RETURN
26100 REM *** LDA (INDIRECT,X)
26110 GOSUB3000
26120 OO=PEEK(O+X)+256*PEEK(O+X+1)
26130 A= PEEK(OO)
26140 N=(A AND 128)/128
26150 Z=-(A=0)
26160 RETURN
26200 REM *** LDX #IMMEDIATE
26210 GOSUB3000
26220 X=O
26230 N=(X AND 128)/128
26240 Z=-(X=0)
26250 RETURN
26400 REM *** LDY ZEROPAGE
26410 GOSUB3000
26420 Y= PEEK(O)
26430 N=(Y AND 128)/128
26440 Z=-(Y=0)
26450 RETURN
26500 REM *** LDA ZEROPAGE
26510 GOSUB3000
26520 A= PEEK(O)
26530 N=(A AND 128)/128
26540 Z=-(A=0)
26550 RETURN
26600 REM *** LDX ZEROPAGE
26610 GOSUB3000
26620 X= PEEK(O)
26630 N=(X AND 128)/128
26640 Z=-(X=0)
26650 RETURN
26800 REM *** TAY
26810 Y=A
26820 Z=-(Y=0)
26830 N=(Y AND 128)/128
26840 RETURN
26900 REM *** LDA #IMMEDIATE
26910 GOSUB3000
26920 A=O
26930 N=(A AND 128)/128
26940 Z=-(A=0)
```

```
26950 RETURN
27000 REM *** TAX
27010 X=A
27020 Z=-(X=0)
27030 N=(X AND 128)/128
27040 RETURN
27200 REM *** LDY ABSOLUTE
27210 GOSUB4000
27220 Y=PEEK(OO)
27230 N=(Y AND 128)/128
27240 Z=-(Y=0)
27250 RETURN
27300 REM *** LDA ABSOLUTE
27310 GOSUB4000
27320 A=PEEK(OO)
27330 N=(A AND 128)/128
27340 Z=-(A=0)
27350 RETURN
27400 REM *** LDX ABSOLUTE
27410 GOSUB4000
27420 X=PEEK(OO)
27430 N=(X AND 128)/128
27440 Z=-(X=0)
27450 RETURN
27600 REM *** CLV
27610 V=0
27620 RETURN
27700 REM *** LDA (INDIRECT),Y
27710 GOSUB3000
27720 OO=PEEK(O)+256*PEEK(O+1)+Y
27730 A=PEEK(OO)
27740 N=(A AND 128)/128
27750 Z=-(A=0)
27760 RETURN
28000 REM *** LDY ZEROPAGE,X
28010 GOSUB3000
28020 Y=PEEK(O+X)
28030 N=(Y AND 128)/128
28040 Z=-(Y=0)
28050 RETURN
28100 REM *** LDA ZEROPAGE,X
28110 GOSUB3000
28120 A=PEEK(O+X)
28130 N=(A AND 128)/128
28140 Z=-(A=0)
28150 RETURN
28200 REM *** LDX ZEROPAGE,Y
28210 GOSUB3000
28220 X=PEEK(O+Y)
28230 N=(X AND 128)/128
28240 Z=-(X=0)
```

```
28250 RETURN
28400 REM *** CLV
28410 V=0
28420 RETURN
28500 REM *** LDA ABSOLUTE,Y
28510 GOSUB4000
28520 A= PEEK(OO+Y)
28530 N=(A AND 128)/128
28540 Z=-(A=0)
28600 REM *** TSX
28610 X=S
28620 Z=-(X=0)
28630 N=(X AND 128)/128
28640 RETURN
28800 REM *** LDY ABSOLUTE,X
28810 GOSUB4000
28820 Y= PEEK(OO+X)
28830 N=(Y AND 128)/128
28840 Z=-(Y=0)
28850 RETURN
28900 REM *** LDA ABSOLUTE,X
28910 GOSUB4000
28920 A= PEEK(OO+X)
28930 N=(A AND 128)/128
28940 Z=-(A=0)
28950 RETURN
29000 REM *** LDX ABSOLUTE,Y
29010 GOSUB4000
29020 X= PEEK(OO+Y)
29030 N=(X AND 128)/128
29040 Z=-(X=0)
29050 RETURN
29200 REM *** CPY #IMMEDIATE
29210 GOSUB3000
29220 C = -((Y-O)>=0)
29230 N = ((Y-O) AND 128)/128
29240 Z = -((Y-O)=0)
29250 RETURN
29300 REM *** CMP (INDIRECT,X)
29310 GOSUB3000
29320 OO=PEEK(O+X)+256*PEEK(O+X+1)
29330 M=PEEK(OO)
29340 GOSUB8400
29350 RETURN
29600 REM *** CPY ZEROPAGE
29610 GOSUB3000
29620 M=PEEK(O)
29630 C = -((Y-M)>=0)
29640 N = ((Y-M) AND 128)/128
29650 Z = -((Y-M)=0)
29660 RETURN
```

```
29700 REM *** CMP ZEROPAGE
29710 GOSUB3000
29720 M=PEEK(O)
29730 GOSUB8400
29740 RETURN
29800 REM *** DEC ZEROPAGE
29810 GOSUB3000
29820 M=PEEK(O)
29830 M=(M-1) AND 255
29840 POKEO,M
29850 Z=-(M=0)
29860 N=(M AND 128)/128
29870 RETURN
30000 REM *** INY
30010 Y=(Y+1) AND 255
30020 Z=-(Y=0)
30030 N=(Y AND 128)/128
30040 RETURN
30100 REM *** CMP #IMMEDIATE
30110 GOSUB3000
30120 M=O
30130 GOSUB8400
30140 RETURN
30200 REM *** DEX
30210 X=(X-1) AND 255
30220 Z=-(X=0)
30230 N=(X AND 128)/128
30240 RETURN
30400 REM *** CPY ABSOLUTE
30410 GOSUB4000
30420 M=PEEK(OO)
30430 C = -((Y-M)>=0)
30440 N = ((Y-M) AND 128)/128
30450 Z = -((Y-M)=0)
30460 RETURN
30500 REM *** CMP ABSOLUTE
30510 GOSUB4000
30520 M=PEEK(OO)
30530 GOSUB8400
30540 RETURN
30600 REM *** DEC ABSOLUTE
30610 GOSUB4000
30620 M=PEEK(OO)
30630 M=(M-1) AND 255
30640 POKEOO,M
30650 Z=-(M=0)
30660 N=(M AND 128)/128
30670 RETURN
30800 REM *** BNE
30810 GOSUB3000
30820 IFZ=0THENGOSUB7000
```

```
30830 RETURN
30900 REM *** CMP (INDIRECT),Y
30910 GOSUB3000
30920 OO=PEEK(O)+256*PEEK(O+1)+Y
30930 M=PEEK(OO)
30940 GOSUB8400
30950 RETURN
31300 REM *** CMP ZEROPAGE,X
31310 GOSUB3000
31320 M=PEEK(O+X)
31330 GOSUB8400
31340 RETURN
31400 REM *** DEC ZEROPAGE,X
31410 GOSUB3000
31420 M=PEEK(O+X)
31430 M=(M-1) AND 255
31440 POKEO+X,M
31450 Z=-(M=0)
31460 N=(M AND 128)/128
31470 RETURN
31600 REM *** CLD
31610 D=0
31620 RETURN
31700 REM *** CMP ABSOLUTE,Y
31710 GOSUB4000
31720 M=PEEK(OO+Y)
31730 GOSUB8400
31740 RETURN
32100 REM *** CMP ABSOLUTE,X
32110 GOSUB4000
32120 M=PEEK(OO+X)
32130 GOSUB8400
32140 RETURN
32200 REM *** DEC ABSOLUTE,X
32210 GOSUB4000
32220 M=PEEK(OO+X)
32230 M=(M-1) AND 255
32240 POKEOO+X,M
32250 Z=-(M=0)
32260 N=(M AND 128)/128
32270 RETURN
32400 REM *** CPX #IMMEDIATE
32410 GOSUB3000
32420 C = -((X-O)>=0)
32430 N = ((X-O) AND 128)/128
32440 Z = -((X-O)=0)
32450 RETURN
32500 REM *** SBC (INDIRECT,X)
32510 GOSUB3000
32520 OO=PEEK(O+X)+256*PEEK(O+X+1)
32530 M=PEEK(OO)
```

```
32540 GOSUB8200
32550 RETURN
32800 REM *** CPX ZEROPAGE
32810 GOSUB3000
32820 M=PEEK(O)
32830 C = -((X-M)>=0)
32840 N = ((X-M) AND 128)/128
32850 Z = -((X-M)=0)
32860 RETURN
32900 REM *** SBC ZEROPAGE
32910 GOSUB3000
32920 M=PEEK(O)
32930 GOSUB8200
32940 RETURN
33000 REM *** INC ZEROPAGE
33010 GOSUB3000
33020 M=PEEK(O)
33030 M=(M+1) AND 255
33040 POKEO,M
33050 Z=-(M=0)
33060 N=(M AND 128)/128
33070 RETURN
33200 REM *** INX
33210 X=(X+1) AND 255
33220 Z=-(X=0)
33230 N=(X AND 128)/128
33240 RETURN
33300 REM *** SBC #IMMEDIATE
33310 GOSUB3000
33320 M=O
33330 GOSUB8200
33340 RETURN
33400 REM *** NOP
33410 RETURN
33600 REM *** CPX ABSOLUTE
33610 GOSUB4000
33620 M=PEEK(OO)
33630 C = -((X-M)>=0)
33640 N = ((X-M) AND 128)/128
33650 Z = -((X-M)=0)
33660 RETURN
33700 REM *** SBC ABSOLUTE
33710 GOSUB4000
33720 M=PEEK(OO)
33730 GOSUB8200
33740 RETURN
33800 REM *** INC ABSOLUTE
33810 GOSUB4000
33820 M=PEEK(OO)
33830 M=(M+1) AND 255
33840 POKEOO,M
```

```
33850 Z=-(M=0)
33860 N=(M AND 128)/128
33870 RETURN
34000 REM *** BEQ
34010 GOSUB3000
34020 IFZ=1THENGOSUB7000
34030 RETURN
34100 REM *** SBC (INDIRECT),Y
34110 GOSUB3000
34120 OO=PEEK(O)+256*PEEK(O+1)+Y
34130 M=PEEK(OO)
34140 GOSUB8200
34150 RETURN
34500 REM *** SBC ZEROPAGE,X
34510 GOSUB3000
34520 M=PEEK(O+X)
34530 GOSUB8200
34540 RETURN
34600 REM *** INC ZEROPAGE,X
34610 GOSUB3000
34620 M=PEEK(O+X)
34630 M=(M+1) AND 255
34640 POKEO+X,M
34650 Z=-(M=0)
34660 N=(M AND 128)/128
34670 RETURN
34800 REM *** SED
34810 D=1
34820 RETURN
34900 REM *** SBC ABSOLUTE,Y
34910 GOSUB4000
34920 M=PEEK(OO+Y)
34930 GOSUB8200
34940 RETURN
35300 REM *** SBC ABSOLUTE,X
35400 REM *** INC ABSOLUTE,X
35410 GOSUB4000
35420 M=PEEK(OO+X)
35430 M=(M+1) AND 255
35440 POKEOO+X,M
35450 Z=-(M=0)
35460 N=(M AND 128)/128
35470 RETURN
35510 GOSUB4000
35520 M=PEEK(OO+X)
35530 GOSUB8200
35540 RETURN
49000 FORHH=34768TO34783:READH:POKEHH,H:NEXT:RETURN
50000 DATA9,240,8,8,88,8,8,8,8,8,8,8,8,0,0,0
```

Das Programm ist wie folgt aufgebaut:

Die Zeilen 10 bis 90 geben die Kopfzeile des Programms aus und beinhalten die Eingaben. Dort werden auch einige Felder und Funktionen definiert, die zu späteren Berechnungen notwendig sind. In Zeile 70 befindet sich der Aufruf eines Unterprogrammes ab Zeile 49000, welches ein Demonstrationsprogramm in die Speicherzellen ab 34768=$87D0 'poked'. In den Zeilen 100 bis 900 sind einige Unterprogramme enthalten, die der Umrechnung von Hexadezimalzahlen dienen, die hier nicht weiter erläutert werden (vgl. Kapitel 5.2 in Band 1). Ab Zeile 1000 bis 1060 befindet sich die Schleife, die für jeden Befehl durchlaufen wird. Dabei wird zunächst der aktuelle Stand der Prozessorregister angezeigt (Unterprogrammaufruf 2000), dann die Variable OP mit dem Wert der momentan über den Programmzähler adressierten Speicherzelle besetzt und anschließend das Unterprogramm ab Zeile 1100 aufgerufen, welches einen Befehl auswertet. Schließlich wird noch der Programmzähler um eins erhöht und wieder zur Zeile 1000 gesprungen, wenn kein Abbruchkriterium erfüllt ist.

Das Unterprogramm ab Zeile 1100 ist ein Sprungverteiler auf die einzelnen Programmstücke, jeweils eine Sprung für jeden Befehl. Ein unzulässiger Befehl bedeutet immer einen Sprung auf das Unterprogramm ab Zeile 1400, welches mit einer entsprechenden Meldung am Bildschirm endet.

Das Unterprogramm ab Zeile 2000 dient zur Anzeige aller Prozessorregister. Das Statusregister wird dabei binär dargestellt, wobei ein Zeichen durch die Funktion CHR$ (48+N) gebildet wird, was hier einer "0" oder "1" entspricht, je nachdem, ob N gleich 0 oder 1 ist. Die anderen Prozessorregister werden mit Hilfe des Unterprogrammes ab Zeile 200 in hexadezimale Zahlen umgewandelt. Diese werden gesammelt in Zeile 2090 in der zweiten Bildschirmzeile ausgeben.

Die Zeilen 3000 bis 9000 enthalten Unterprogramme, die für die Ausführung von Befehlen notwendig sind, bzw. allgemeine Unterprogramme, die die Programmierung der anderen vereinfachen.

Ab Zeile 3000 wird ein 1-Byte-Operand geholt, was dadurch geschieht, daß der Programmzähler um 1 erhöht wird und der Variablen O der Wert der angesprochenen Speicherzelle zugewiesen wird. O soll hier die Abkürzung für 1 Byte-Operand sein. Im Unterprogramm ab 4000 wird entsprechend ein 2-Byte-Operand geholt, der in der Variablen OO abgelegt wird.

Die Unterprogramme ab den Zeilen 5000 bis 6000 fassen die einzelnen Statusflags zur Variablen P zusammen bzw. zerlegen diese.

Ab Zeile 7000 steht die Ausführung eines relativen Sprunges.

Da die Addition ein komplizierterer Prozess ist, da insbesondere Binär- oder Dezimalarithmetik verwendet werden können, wird diese in dem Unterprogramm ab Zeile 8000 behandelt. Analog wird ab 8200 die Subtraktion behandelt. In diesen Unterprogrammen wird zunächst erfragt, ob das Dezimalflag gesetzt ist. Wenn nicht, ergibt sich der Neue Wert des Akkumulators einfach aus:

Alter Wert + Speicherwert + Carry-Flag.

Das neue Carry-Flag wird dann gesetzt, wenn der Wert des Akkumulators größer als 255 ist. Dann wird der Akkumulator noch auf ein Wert bis 255 zurückgesetzt (weil die Variable A, anders als der Akkumulator, Zahlen größer als 255 aufnehmen kann) und das Negativ- und das Zero-Flag, also hier die Variablen N und Z entsprechend besetzt. Wenn jedoch das Dezimal-Flag gesetzt ist, so werden zuerst die niederwertigen 4 Bit des Akkumulators mit den niederwertigen 4 Bit des Speichers verknüpft und anschließend die höherwertigen 4 Bit. Ein Übertrag kommt immer dann zustande, wenn das Ergebnis größer als 9 ist.

Ab Zeile 10000 stehen die Unterprogrammstücke, die jeweils einem Maschinenbefehl entsprechen. Die Zeilennummern der Unterprogramme sind dabei wie folgt aufgebaut:

Zeilennummer Befehl = 10000 + 100 x Wert Maschinencode.

Da wir an dieser Stelle davon ausgehen, daß Basic hinlänglich bekannt ist, werden für die Unterprogramme zu den einzelnen Befehlen keine Erklärungen mehr gegeben. Vielmehr sollen gerade die Unterprogramme in Basic dazu dienen, Ihnen die Arbeitsweise der Befehle zu verdeutlichen.

DEMO6510 **10 - 50000**

Variablen:

Name	Typ	Bereich	Bedeutung
A	G	0...255	Akkumulator
B	G	0,1	Break-Flag
C	G	0,1	Carry-Flag
D	G	0,1	Dezimal-Flag
H	H	Integer	Hilfsvariable
H$	H	2 Zeichen	Hilfsvariable
HO$	H	31 Zeichen	Eingabezeile
HE$	G	'0123456789ABCDEF'	Hexadezimalziffern
HH	H	0...65535	Hilfsvariable
HH$	H	4 Zeichen	Hilfsvariable
HI	H/R	0...255	Höherwertiges Byte
HP	H	0...10	Anzahl Abbruchpunkte
I	G	0,1	Interrupt-Disable-Flag
J	H	Integer	Laufvariable
LO	H/R	0...255	Niederwertiges Byte
M	H	0...65535	Speicheradresse
N	G	0,1	Negativ-Flag
O	H	0...255	Operand (1 Byte)
OO	H	0...65535	Operand (2 Byte)
OP	H	0...255	Befehlscode
P	G	0...255	Prozessorstatus
P5	G	0,1	Bit 5 von P
PC	G	0...65535	Programmzähler
S	G	0...255	Stapelzeiger
V	G	0,1	Overflow-Flag
X	G	0...255	X-Register
Y	G	0...255	Y-Register
Z	G	0,1	Zero-Flag

Felder (Arrays):

Name	Dimen.	Typ	Bereich	Bedeutung
HP	10	G	0...65535	Abbruchstellen
ST%	255	G	0...255	Kellerspeicher

Unterprogrammaufrufe : (nur im Bereich 10 - 8440)

in	nach	Zweck
22	2000	Anzeige Register
26	400	Wert von 4-stelliger Hexzahl bestimmen
44	350	Wert von 2-stelliger Hexzahl bestimmen
46	350	Wert von 2-stelliger Hexzahl bestimmen
48	350	Wert von 2-stelliger Hexzahl bestimmen
50	350	Wert von 2-stelliger Hexzahl bestimmen
62	400	Wert von 4-stelliger Hexzahl bestimmen
70	49000	Beispielprogramm 'poken'
270	200	2-stellige Hexzahl bilden
300	200	2-stellige Hexzahl bilden
420	350	Wert von 2-stelliger Hexzahl bestimmen
450	350	Wert von 2-stelliger Hexzahl bestimmen
1010	2000	Anzeige Register
1045	200	2-stellige Hexzahl bilden
1050	1100	Befehl ausführen
2010	250	4-stellige Hexzahl bilden
2040	200	2-stellige Hexzahl bilden
2050	200	2-stellige Hexzahl bilden
2060	200	2-stellige Hexzahl bilden
2070	200	2-stellige Hexzahl bilden

Verzweigungen nach außen :

in Ze	nach	Bedingung	Bemerkung
1090	STOP	PC = HP(J)	Abbruchpunkt err.
1370	STOP	OP größer als 255	
1410	STOP		illegaler Befehl

1.9 Zusätzliche und illegale Befehle

Wie Sie aus Kapitel 1.7 ersehen konnten, sind nicht alle der 256 theoretisch möglichen Operations-Codes mit tatsächlichen Funktionen belegt. Sie können jedoch dem Prozessor einen solchen Operations-Code anbieten. Die Reaktion des Prozessors auf einen solchen Befehl ist durch

seine interne Struktur definiert. Bei einigen illegalen Befehlen, hört der Prozessor auf zu arbeiten und kann dann nur noch mit einem Reset-Impuls reaktiviert werden. Dann ist natürlich der momentane Programmzähler verloren. Einige Befehle führen jedoch nicht zum Absturz des Rechners, sondern führen exotische Funktionen aus, die eine Kombination von offiziellen Operationen darstellen. So z.B. die folgenden drei Befehle:

A7 aa Der Akkumulator und das X-Register werden mit dem Inhalt der Zero-Page-Adresse aa geladen.

87 aa Das Ergebnis einer UND-Funktion zwischen Akku und X-Register wird in die Zero-Page-Adresse aa gespeichert.

97 aa Das Ergebnis einer UND-Funktion zwischen Akku und X-Register wird in die Zero-Page-Adresse aa+Y gespeichert.

Es gibt sicher weitere inoffizielle Befehle, das Austesten anderer Möglichkeiten wollen wir dem Leser überlassen. Sie können sich kleine Testprogramme schreiben und mit etwas Mühe und Geduld sich die Funktion sämtlicher inoffiziellen Befehle, die nicht zum Absturz des Prozessors führen, herausfinden.

Es ist jedoch zu beachten, daß die Prozessoren mittlerweile geändert worden sein könnten. Da es sich eben um inoffizielle Befehle handelt, muß die Herstellerfirma nicht darauf acht geben, ob deren Funktion unverändert bleibt. Deshalb ist es auch ratsam, auf Ihrem eigenen Rechner die oben angebenenen Befehle zuerst auszuprobieren, bevor Sie diese einsetzen.

Eine weitere Besonderheit sei noch angemerkt: folgt auf einen indirekten Sprung (Operations-Code $6C) das Byte $FF, so wird der Sprung falsch ausgeführt. Wenn der Befehl z.B. heißen sollte 6C FF C0, so wird die Sprungadresse nicht aus den Zellen $C0FF und $C100, sondern aus $C0FF und $C000 geholt. Dies ist ein Fehler im Prozessor, der im Moment noch existiert. Es kann sein, daß vom Prozessor ein neuer Typ herausgebracht wird, der diesen Fehler nicht mehr aufweist.

2

Zusammenarbeit von Maschinenprogrammen mit Basic

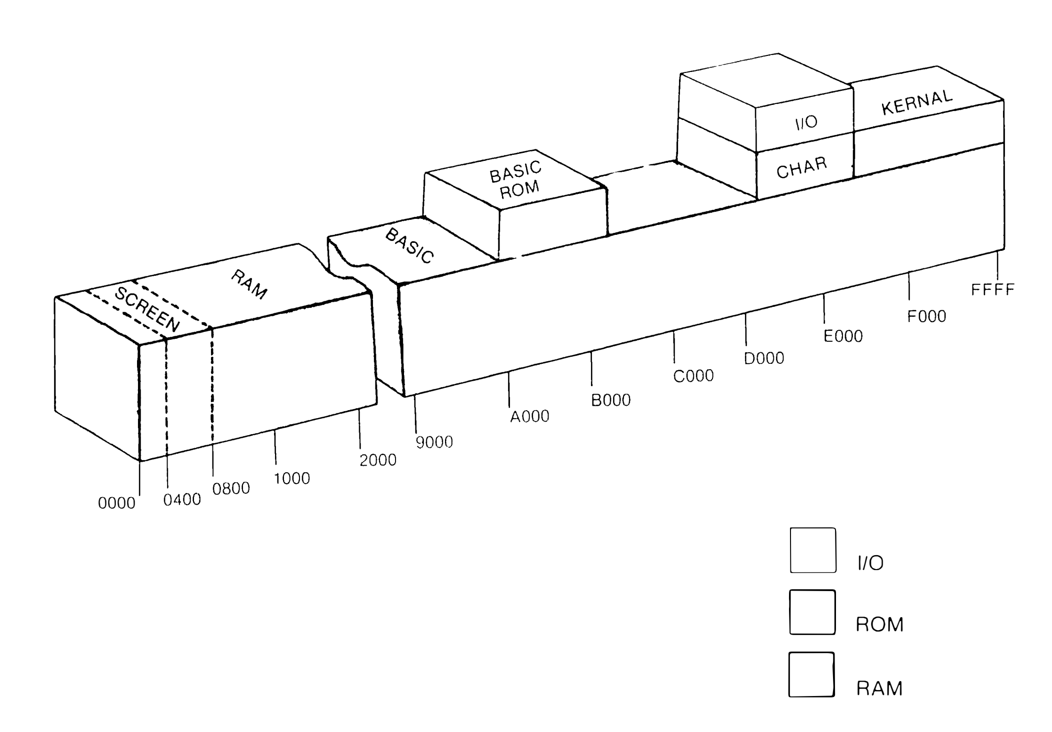
2. Zusammenarbeit von Maschinenprogrammen mit Basic

In diesem Kapitel soll der Zusammenhang zwischen Maschinenprogrammen und Basic sowie dem Betriebssystem dargestellt werden. Dazu gehen wir zunächst auf die Speichereinteilung des Commodore 64 ein. Dann werden die Zahldarstellungen (Integer und Fließkomma) behandelt, sowie die Variablenhandhabung innerhalb von Basic-Programmen.

Der nächste Abschnitt ist den ROM-Routinen (Basic und Kernal) vorbehalten. Daran schließt sich ein Kapitel an, daß die Möglichkeiten des Ladens von Maschinenprogrammen aufzeigt (LOAD-Befehl mit Merker).

Die nächsten beiden Kapitel widmen sich dem Einbinden von Maschinenprogrammen in Basic, und den Abschluß bildet eine Übersicht über die Zero-Page.

2.1 Speicheraufteilung im Commodore 64



Der Commodore 64 besitzt 64 KByte Schreib-/Lesespeicher (RAM), außerdem kann er 20 KByte Nur-Lesespeicher (ROM) adressieren, die das Basic, Betriebssystem und den normalen Zeichensatz enthalten. Schließlich kann er noch einen 4-KByte-Block adressieren, welcher die Ein-/Ausgabe-Bausteine enthält.

Der 6510 kann jedoch nur 64 KByte adressieren, da er nur 16 Adressleitungen besitzt. Deshalb werden seine Ausgabeleitungen P0-P2 benützt, um zwischen ROM, RAM und I/O Bereichen umzuschalten, die Ausgabeanschlüsse P3-P5 werden für den Kassettenrecorder benützt.

Hier die Tabelle mit den Standardbelegungen der ein Ausgabeleitungen des 6510:

Name	Bit	Richtung	Beschreibung
LORAM	0	Ausgabe	Schalter für Basic-RAM/ROM ($A000 - $BFFF)
HIRAM	1	Ausgabe	Schalter für KERNAL-RAM/ROM ($E000 - $FFFF)
CHAREN	2	Ausgabe	Schalter für I/O-ROM ($D000 - $DFFF)
	3	Ausgabe	Schreibleitung für Kassettenrekorder
	4	Eingabe	Zeigt gedrückte Taste an Kassettenrekorder an
	5	Ausgabe	Motorsteuerung Kassette

Platz für Tabelle

LORAM

Dieser Anschluß dient zum Ausblenden des 8 KByte Basic-ROMS. Liegt dieser Anschluß auf '1', so werden bei einer Lese-Operation die Werte aus dem ROM gelesen, bei einer Schreib-Operation die Werte im RAM abgelegt. Liegt die Leitung auf '0', so werden sowohl Schreib-und Lese-Operationen mit dem RAM durchgeführt.

HIRAM

Diese Leitung blendet das Betriebssystem-ROM (KERNAL) ein und aus. Liegt diese Leitung auf '1', so wirken die Lese-Operationen auf das ROM, die Schreib-Operationen auf das RAM. Liegt die Leitung auf '0', so erscheint nur das RAM für beide Transferrichtungen.

CHAREN

Diese Leitung wird benützt, um den Zeichengenerator auslesen zu können. Dieser befindet sich im Adressbereich von $D000 bis $DFFF. Im gleichen Adressbereich befinden sich auch die Ein-/Ausgabe-Einheiten. Ist die Leitung CHAREN gleich '1', so erscheinen folgende I/O-Bausteine:

```
!=============!
! VIC         !  $D000 / 53248
!             !
!             !
!             !  $D3FF
!-------------!
! SID         !  $D400 / 54272
!             !
!             !
!             !  $D7FF
!-------------!
! (Farb-RAM)  !  $D800 / 55296
!             !
!             !
!             !  $DBFF
!-------------!
! CIA 1       !  $DC00 / 56320
!-------------!
! CIA 2       !  $DD00 / 56576
!-------------!
! I/O Erw. 0  !  $DE00 / 56832
!-------------!
! I/O Erw. 1  !  $DF00 / 57088
!-------------!
```

Ist die Leitung CHAREN gleich '0', so werden bei einer Lese-Operation die Daten aus den Zeichengenerator-ROM gelesen, bei einer Schreib-Operation die Daten im RAM abgelegt. Diese im RAM abgelegten Daten können jedoch nicht wieder gelesen werden, deshalb ist dieser RAM-Bereich für den Anwender nutzlos.

Das Umschalten dieser Leitung darf nur in einem Maschinenprogramm erfolgen, welches Unterbrechungen verhindert, solange die Leitung auf '0' gehalten wird.

2.2 Zahldarstellung

Im Betriebssystem und in den anderen Programmen kommen die verschiedensten Möglichkeiten vor, verschiedenartige Zahlen darzustellen. Der Prozessor kann natürlich immer nur mit einzelnen Bytes rechnen, also Kombinationen von 8 mal '0' und '1'. Es kommt daher darauf an, wie wir eine gegebene Kombination interpretieren, um sie einem Zahlwert zuzuordnen.

Wir wollen uns im folgenden die für Mikrocomputer gebräuchlichsten Darstellungsarten ansehen.

8-Bit vorzeichenlose Darstellung

Hier haben die 8 Bit eine Speicherzelle folgende Wertigkeiten:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
128	64	32	16	8	4	2	1

Mit dieser Darstellung können wir Werte von 0 bis 255 darstellen. Dies wird z.B. angewendet bei der Interpretation von einfachen Zählern, so z.B. dem X-Register.

8-Bit vorzeichenbehaftete Darstellung

Hier werden folgende Wertigkeiten benützt:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-128	64	32	16	8	4	2	1

Der darstellbare Zahlenbereich liegt dann zwischen -128 und +127. Diese Interpretation wird z.B. bei der Ausführung von relativen Sprüngen (siehe auch Kapitel 1.4) verwendet. Außerdem erklärt diese Darstellung den Sinn des Negativ-Flags, das dann gesetzt wird, wenn ein Ergebnis das Bit 7 gesetzt hat, also es nach dieser Darstellung

negativ ist. Außerdem kann man das Overflow-Flag erklären, das dann gesetzt wird, wenn bei einer Addition oder Subtraktion der zulässige Bereich -128 bis +127 überschritten wird.

16-Bit vorzeichenlose Darstellung

Diese Darstellung ähnelt der 8-Bit vorzeichenlosen Darstellung, jedoch werden hier 2 aufeinanderfolgende Bytes verwendet. Bit 0 bis 7 haben die gleichen Wertigkeiten wie bei der 8-Bit vorzeichenlosen Darstellung, Bit 8 bis 15 die folgenden:

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
32768	16384	8192	4096	2048	1024	512	256

Der darstellbare Bereich reicht von 0 bis 65535. Damit werden im allgemeinen Adressdaten gespeichert. Natürlich ist diese Darstellung auch für ganze Zahlen geeignet.

16-Bit vorzeichenbehaftete Darstellung

Die Bits 0 bis 7 haben die gleichen Wertigkeiten wie bei 8-Bit vorzeichenloser (!) Darstellung. Die Wertigkeiten der Bits 8 bis 15 sind die folgenden:

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
-32768	16384	8192	4096	2048	1024	512	256

Der darstellbare Bereich liegt zwischen -32768 und +32767. Mit dieser Darstellung werden im Rechner sogenannte INTEGER-Zahlen dargestellt werden, wie z.B. die %-Variablen.

Fließkommadarstellung

Wir wollen hier die Art der Fließkommadarstellung besprechen, wie sie beim Commodore für die Speicherung von gewöhnlichen Variablen verwendet wird. Damit kann ein Bereich von etwa -1 E38 bis +1 E38 abgedeckt werden. Diese Darstellung benötigt 5 Byte. Das erste Byte beinhaltet den sogenannten Exponenten, die restlichen vier die sogenannte Mantisse. Das Vorzeichen der Mantisse ist im Höchstwerti-

gen Bit dieser vier Bytes gespeichert. Dabei bedeutet '1' negativ und '0' positiv.

Der Exponent ist um $80 erhöht abgespeichert. Der um $80 verminderte Wert gibt dann an, wie oft man die Mantisse mit 2 multplizieren muß, damit man den Zahlwert erhält. Ein Exponent $7F bedeutet, das man die Mantisse durch 2 dividieren muß. Die Zahl Null wird durch ein Exponenten $00 dargestellt.

Die Wertigkeiten der 32 Bit in der Mantisse sind wie folgt zu verstehen:

(Bit 31)	Bit 30	Bit 29	Bit 28	......	Bit 0
0.5	0.25	0.125	0.0625	......	2 hoch minus 32

Die Fließkammadarstellung mit Mantisse und Exponent ist jedoch so noch nicht eindeutig. Man kann z.B. die Dezimalzahl 0.25 wie folgt darstellen:

$ 80 40 00 00 00

oder

$ 82 10 00 00 00

oder

$ 83 08 00 00 00

bzw. die ersten beiden Byte binär

1000 0000 0100 0000

oder

1000 0010 0001 0000

oder

1000 0011 0000 1000

Im Speicher werden Fließkammazahlen immer normiert abgespeichert, d.h. so, daß das höchstwertigste Bit der Mantisse, welches die Wertigkeit 0.5 besitzt, stets auf '1' gesetzt ist. Weil aber dieses Bit immer gesetzt sein müsste, hat es kein Informationswert und wird deshalb nicht abgespeichert, sondern an dessen Stelle wird das Vorzeichen gespeichert. Im folgenden sehen Sie eine kleine Tabelle von Fließkommazahlen und deren hexadezimaler Darstellung.

Speicher-Format	Dezimal
84 80 00 00 00	- 8
83 C0 00 00 00	- 6
82 80 00 00 00	- 2
81 C0 00 00 00	- 1,5

Speicher-Format	Dezimal
81 80 00 00 00	- 1
80 80 00 00 00	- 5
7F.80 00 00 00	- .25
00 00 00 00 00	0
7F 00 00 00 00	.25
80 00 00 00 00	.5
81 00 00 00 00	1
81 40 00 00 00	1.5
82 00 00 00 00	2
82 40 00 00 00	3
83 00 00 00 00	4
84 00 00 00 00	8

6-Byte-Fließkommazahlen (Register-Format)

Im Fließkomma-Akkumulator, einem Speicherbereich in der Zero-Page, werden die Fließkommazahlen mit 6 Byte gespeichert. Dabei gleicht das Format im wesentlichen den Fließkommazahlen im oben beschriebenen Speicherformat. Das Vorzeichen ist im 6. Byte abgespeichert. Das Höchstwertigste Bit des 2. Byte, also das erste Bit der Mantisse, ist immer gesetzt (Wert 0.5). Die Zahl 1 würde wie folgt ausseßen: $ 81 80 00 00 00 00. Auf das Register-Format soll hier nicht näher eingegangen werden, weil nur der Rechner diese Darstellung benötigt.

2.3 Variablen im Commodore-Basic

Es existieren drei Speicherbereiche, die für die Speicherung der Variablen verwendet werden. Der erste enthält die einfachen Variablen, der zweite die Felder und im dritten werden die Inhalte von String-Variablen (also die Zeichenreihen selbst) gespeichert. Zur Abgrenzung dieser drei Bereiche dienen die folgenden Zeiger in der Zero-Page:

Bezeichnung	Hex.	Dezimal	Bedeutung
STARTVAR	2D,2E	45,46	Start der Variablen
STARTARR	2F,30	47,48	Start der Arrays
ENDARR	31,32	49,50	Ende der Arrays
STARTSTR	33,34	51,52	Beginn der Zeichenreihen
ENDRAM	37,38	55,56	Basic-RAM Ende

Zwischen STARTVAR und STARTARR sind die einfachen Variablen gespeichert, die Felder zwischen STARTARR und ENDARR. Die Zeichenreihen werden im Bereich zwischen STARTSTR und ENDRAM abgelegt.

Datenformate

Die Fließkommazahlen werden durch 5 Byte dargestellt, Integerzahlen durch 2 Byte (vgl. dazu Kapitel 2.2).

Stringvariable werden als 3 Byte abgespeichert. Das erste enthält die Länge der Strings (0-255), die nächsten beiden enthalten einen Zeiger (Adresse) auf den Beginn der Zeichenreihe. Die Zeichenreihe selbst steht dann im Bereich zwischen STARTSTR und ENDRAM oder im Basic-Programmtext (bei Zuweisung einer String-Konstanten oder Verwendung von DATA und READ).

Name und Typ von Variablen

Der Name und der Typ einer Variablen werden grundsätzlich in 2 Byte dargestellt. Diese 2 Byte sind die ASCII-Codes der beiden Buchstaben des Variablennamens. Dabei gibt es jedoch folgende Besonderheiten: Ein nicht vorhandenes Zeichen (bei einem Variablennamen, der nur aus einem Buchstaben besteht) wird durch den Code 0 dargestellt. Bei Stringvariablen wird das höchstwertigste Bit des zweiten Bytes gesetzt. Bei Integervariablen wird bei beiden Bytes das höchstwertigste Bit gesetzt.

Einfache Variable

Einfache Variable werden nacheinander in 7-Byte-Feldern abgelegt. Dort stehen jeweils 2 Byte für den Variablennamen und dann 5 Byte für den Variableninhalt. Bei Integer- und String-Variablen werden die 5 Byte, die für die Daten reserviert sind natürlich nicht voll ausgenützt, sie werden dann mit Nullen belegt.

Felder (Arrays)

Ein Feld besteht jeweils aus einem Feldkopf und anschliessend den einzelnen indizierten Variablen.

Der Feldkopf ist wie folgt aufgebaut: Die ersten beiden Bytes enthalten den Namen und den Typ des Feldes (s.o.). Die nächsten beiden Bytes enthalten die gesamte Länge dieses Feldes; damit kann man die Startadresse des näch-

sten Feldes berechnen.

Byte 5 enthält die Anzahl der Dimensionen des Feldes. Im allgemeinen wird diese Zahl zwischen 1 und 5 liegen.

Danach folgen je 2 Byte, die die Anzahl der Elemente pro Dimension angeben. Wenn ein Feld nur eine Dimension hat ist der Kopf genau 7 Byte lang.

2.4 ROM-Routinen des Commodore 64

Sie können sich die Programmierung von eigenen Maschinenroutinen sehr erleichtern, wenn Sie bereits im ROM eingebaute Routinen mitverwenden. Dabei gibt es im wesentlichen zwei verschiedene Arten von eingebauten ROM-Routinen. Als erstes wären die Basic-Routinen zu nennen, das sind die Routinen zur Variablenverwaltung, Fließkomma-Arithmetik, zur Behandlung von Basic-Fehlermeldungen und ähnlichem. Diese Routinen werden Sie also vor allem dann verwenden, wenn Sie eng mit Basic zusammenarbeiten wollen und insbesondere auch Parameter von oder an Maschinenprogramme durch Basic-Variablen übergeben wollen.

Die andere Gruppe besteht aus den Betriebssystem-Routinen, hier den sogenannten KERNAL-Routinen. Diese Routinen beschäftigen sich mit der Ein/Ausgabe, mit der Bildschirmverwaltung, dem seriellen Bus und ähnlichem. Die Parameterübergabe an diese Routinen ist wesentlich besser normiert, als bei den Basic-Routinen. Das ist durchaus verständlich, weil auf dieses Betriebssystem auch dann zugegriffen werden kann und muß, wenn der Basic-Interpreter ausgeschaltet ist und eine andere Programmiersprache geladen ist oder Sie selbst sich vollkommen von Basic lösen und nur die Ein- und Ausgabe mit eingebauten Routinen durchführen wollen.

Basic-Routinen

Im folgenden wollen wir eine Tabelle der ca. 150 wichtigsten Basic-Routinen abbilden, aus der Sie die Aufrufadresse, die Parameter sowie deren Bedeutung entnehmen können. Weiterhin wird ein gebräuchlicher Name mitangegeben. Beispiele dazu finden Sie im Kapitel 4, wo zahlreiche nützliche Routinen beschrieben sind.

Adresse Hex.	Dez.	üblicher Name	Bemerkungen (A = Ausgabeparameter) (E = Eingabeparameter)
A38A	41866	STAPSUCH	Stapelsuchroutine für FOR...NEXT und GOSUB
A3B8	41912	BLOCKMOV	Blockverschieberoutine E: $5F/$60 Alter Blockanfang
			E: $5A/$5B Altes Blockende+1; $5F/$60 Neues Blockende+1
A3FB	41979	STAPVOLL	prüft auf Platz im Stapel
			E: Halbe Zahl der erforderlichen Bytes im Akku
A408	41992	MAKESPC	schafft Platz im Speicher für neue Prg.zeilen u. Variable
			E: A/Y Adresse, bis zu der Platz benötigt wird
A435	42037	ERROUTM	Ausgabe von 'out of memory'
A437	42039	ERROR	Fehlermeldung ausgeben E: Fehlernummer im X-Register
A474	42823	READY	'READY.' ausgeben und Eingabe erwarten
A480	42112	INLOOP	Eingabe-Warteschleife
A49C	42140	PRGZLE	Löschen und Einfügen von Programmzeilen
A4A9	42153	PRGZLOE	Löschen einer Programmzeile
A533	42291	PRGZBIND	Basic-Programmzeilen neu binden
A560	42336	GETZEI	holt eine Zeile in den Eingabepuffer ab $0200
A571	42353	ERRSTRTL	Ausgabe von 'string too long'
A579	42361	PRGZUM	Umwandlung einer Programm-Zeile im Puffer ab $0200
			in den Interpreterkode
A613	42515	GETPZADR	Startadresse einer Basic-Zeile suchen
			E:$14,$15 Zeilennr.;A:$5F,$60 Adresse;Carry=1,wenn gef.
A642	42562	BBNEW	Basic-Befehl NEW
A65E	42590	BBCLR	Basic-Befehl CLR
A68E	42638	PRGBEGIN	Programmzeiger auf Basic-Start setzen
A69C	42652	BBLIST	Basic-Befehl LIST
A717	42775	KLARTEXT	Interpreterkode in Befehlswort umwandeln und ausgeben
A742	42818	BBFOR	Basic-Befehl FOR
A7AE	42926	INTLOOP	Interpreterschleife führt Basic-Befehle aus;
			prüft auf STOP-Taste; setzt u.a. Zeiger für CONT
A7ED	42989		führt einen Basic-Befehl aus; E:Interpreterkode im Akku

Adresse Hex.	Dez.	üblicher Name	Bemerkungen (A = Ausgabeparameter) (E = Eingabeparameter)
A81D	43037	BBRESTOR	Basic-Befehl RESTORE
A82C	43052	CHSTOP	prüft auf STOP-Taste und bricht evtl. Programm ab
A82F	43055	BBSTOP	Basic-Befehl STOP
A831	43057	BBEND	Basic-Befehl END
A857	43095	BBCONT	Basic-Befehl CONT
A871	43121	BBRUN	Basic-Befehl RUN
A883	43139	BBGOSUB	Basic-Befehl GOSUB
A8A0	43168	BBGOTO	Basic-Befehl GOTO
A8F8	43256	BBRETURN	Basic-Befehl REM
A8F8	43256	BBDATA	Basic-Befehl DATA
A906	43270	NEXTST	sucht nächste Anweisung im Basic-Programm
A909	43273	NEXTLIN	sucht nächste Zeile des Basic-Programms
A928	43304	BBIF	Basic-Befehl IF
A93B	43323	BBREM	Basic-Befehl REM
A94B	43339	BBON	Basic-Befehl ON
A96B	43371	BASZADR	sucht Adresse einer Basic-Zeile
A9A5	43429	BBLET	Basic-Befehl LET
A9BA	43450	WERTZUW	Wertzuweisung an beliebige Variable; E: Var.adr $49/$4A; Typ-Flag (String/num.) im Akku; Integerflag im Stack
A9C4	43460	WERTZUWI	Wertzuweisung an Integer-Variable E: Var.adr $49/$4A
A9D6	43478	WERTZUWR	Wertzuweisung an Real-Variable E: Var.adr $49/$4A
AA2C	43564	WERTZUWS	Wertzuweisung an String-Variable E: Var.adr $49/$4A Adresse des Stringdeskriptors (Lge,AdrL,AdrH) in $64/$65
AA80	43648	BBPRINT1	Basic-Befehl PRINT#
AA86	43654	BBCMD	Basic-Befehl CMD
AAA0	43680	BBPRINT	Basic-Befehl PRINT
AB1E	43806	STROUT	String ausgeben
AB3B	43835	SPCOUT	Ein Leerzeichen bzw. Cursor right (wenn $13=0) ausgeben
AB3F	43839	SPOUT	Ein Leerzeichen ausgeben

Adresse Hex.	Dez.	üblicher Name	Bemerkungen (A = Ausgabeparameter) (E = Eingabeparameter)
AB3E	43838	OUTCURI	Ein Zeichen 'Cursor rechts' ausgeben
AB3F	43839	OUTFRAG	Ein Fragezeichen ausgeben
AB4D	43853	EINFEHL	Fehlerbehandlung bei Eingabe
AB7B	43899	BBGET	Basic-Befehl GET
ABA5	43941	BBINPUT1	Basic-Befehl INPUT#
ABA0	43936	BCLRCH	beendet Eingabe durch Aufruf von CLRCH und setzt $13 zur.
ABBF	43967	BBINPUT	Basic-Befehl INPUT
AC06	44038	BBREAD	Basic-Befehl READ
AD1D	44317	BBNEXT	Basic-Befehl NEXT
AD8A	44426	FRMNUM	holt Ausdruck und prüft auf numerischen Wert Wert steht anschließend im Fließkommaakkumulator (FAC)
AD8D	44429	CHNUM	prüft auf numerisch
AD8F	44431	CHSTR	prüft auf String
AD99	44441	ERRTYPM	Ausgabe von 'type mismatch'
AD9E	44446	FRMEVL	holt und wertet beliebigen Ausdruck aus; Wert steht im FAC (num.) oder kann mit FRESTR (str.) geholt werden
AE83	44675	GETARI	arithmetischen Ausdruck holen
AEA8	44712	FPI	Fliesskommakonstanate Pi
AED4	44756	BBNOT	Basic-Befehl NOT
AEF1	44785	FRMKLA	holt Ausdruck in Klammern
AEF7	44791	CHKKLAZU	prüft, ob 'Klammer zu' im Basic-Text folgt
AEFA	44794	CHKKLAUF	prüft, ob 'Klammer auf' folgt
AEFD	44797	CHKCOM	prüft, ob 'Komma' folgt
AEFF	44799	CHKZEI	prüft, ob Zeichen im Akku im Basic-Text folgt
AF08	44808	ERRSYNT	Ausgabe von 'syntax error'
AF28	44840	VARWERT	holt Wert von Variable aus Basic-Text
AFE6	45030	BBOR	Basic-Befehl OR
AFE9	45033	BBAND	Basic-Befehl AND
B016	45078	VERGL	Vergleichsoperation; Ergebnis steht im FAC

Adresse Hex.	Dez.	üblicher Name	Bemerkungen (A = Ausgabeparameter) (E = Eingabeparameter)
B081	45185	BBDIM	Basic-Befehl DIM
B08B	45195	VARSUC	Variable aus Basic-Text suchen; A: Variablenadr. $47,$48
B0E7	45287	NAMSUC	Sucht Variable; E:Name $45,$46; A: Variablenadr. $47,$48
B113	45331	CHLETT	prüft auf Buchstabe; A:Carry=1,dann Buchstabe
B194	45460	ARRBEG	berechnet Zeiger auf erstes Arrayelement
B1A5	45477	FM32768	Fliesskommakonstante -32768
B1AA	45482	FLPAY	FAC nach Integer wandlen; A: Akku/Y = High/Low
B245	45637	ERRBADS	Ausgabe von 'bad subscipt'
B248	45640	ILLQERR	Ausgabe von 'illigal quantity'
B34C	45900	ARRSIZ	berechnet Arraygrösse
B37D	45949	BBFRE	Basic-Befehl FRE
B39E	45982	BBPOS	Basic-Befehl POS
B3A6	45990	CHDIREKT	Test auf Direkt-Modus, wenn ja -µ 'illegel direct error'
B3AB	45995	ERRILLD	Ausgabe von 'illegal direct'
B3AE	45998	ERRUNFN	Ausgabe von 'undef'd function'
B3B3	46003	BBDEF	Basic-Befehl DEF
B3E1	46049	CHFNSYN	FN-Syntax prüfen
B3F4	46068	BBSTR	Basic-Befehl STR$
B475	46197	STRZEIG	Stringverwaltung, Zeiger auf String berechnen
B487	46215	NEUSTR	Neuen String einrichten
B526	46374	GARCOL	Garbage Collection, nichtgebrauchte Strings entfernen
B63D	46653	STRPLUS	Stringverknüpfung
B643	46659	FRESTR	Stringverwaltung FRESTR
B6EC	46828	BBCHR	Basic-Befehl CHR$
B700	46848	BBLEFT	Basic-Befehl LEFT$
B72C	46892	BBRIGHT	Basic-Befehl RIGHT$
B737	46903	BBMID	Basic-Befehl MID$
B77C	46973	BBLEN	Basic-Befehl LEN
B782	46978	STRPAR	Stringparameter holen

Adresse Hex.	Dez.	üblicher Name	Bemerkungen (A = Ausgabeparameter) (E = Eingabeparameter)
B78B	46987	BBASC	Basic-Befehl ASC
B79B	4703	GETBYT	Liest Byte-Wert aus Basic-Text; A: X
			*: Diese Adresse beinhaltet CHKCOM
B7EB	47083	GETAB	Kombi-Routine: CHKCOM & FRMNUM & FACADR & GETBYT
B7F7	47045	FACADR	Wandelt FAC in Adressformat; A: $14,$15
B7AD	47021	BBVAL	Basic-Befehl VAL
B80D	46861	BBPEEK	Basic-Befehl PEEK
B824	47140	BBPOKE	Basic-Befehl POKE
B82D	47149	BBWAIT	Basic-Befehl WAIT
B849	47177	FACPLU05	FAC = FAC + 0,5
B850	47184	FKMINUS	FAC = Konstante (A/Y) - FAC
B853	47187	FAMINUS	FAC = ARG - FAC
B867	47207	FPLUSK	Plus FAC = Konstante (A/Y) - FAC
B86A	47210	FPLUSA	Plus FAC = ARG + FAC
B97E	47486	ERROVFL	Ausgabe von 'overflow'
B9BC	47548	FKLOG	Fliesskommakonstanten für LOG
B8EA	47338	BBLOG	Basic-Befehl LOG
BA28	47656	FMALK	Multiplikation FAC = Konstante (A/Y) *FAC
BA2B	47659	FMALA	Multiplikation FAC = ARG * FAC
BA8C	47756	FLDARGK	ARG = Konstante (A/Y)
BAE2	47842	FMAL10	FAC = FAC * 10
BAF9	47865	FK10	Fliesskommakonstante 10
BAFE	47870	FDURCH10	FAC = FAC / 10
BB0F	47887	FKDIV	FAC = Konstante (A/Y) / FAC
BB12	47810	FADIV	FAC = ARG / FAC
BB8A	48010	DIVZERR	Ausgabe von 'division by zero
BBA2	48034	FLDFACK	FAC = Konstante (A/Y)
BBC4	48068	FACA4	Akku#4 = FAC
BBCA	48074	FACA3	Akku#3 = FAC

Adresse Hex.	Dez.	üblicher Name	Bemerkungen (A = Ausgabeparameter) (E = Eingabeparameter)
BBD0	48080	FACVAR	Variable = FAC
BBFC	48119	ARGFAC	FAC = ARG
BC0C	48140	FACARG	ARG = FAC
BC1B	48155	FACRUND	FAC runden
BC2B	48171	FSGN	Vorzeichen von FAC holen
BC39	48185	BBSGN	Basic-Befehl SGN
BC58	48216	BBABS	Basic-Befehl ABS
BC5B	48219	FVGLAY	Konstante (A/Y) mit FAC vergleichen
BC9B	48283	FLPINT	Umwandlung FAC nach Integer
BCCC	48332	BBINT	Basic-Befehl INT
BCF3	48371	ASCFLP	Umwandlung ASCII nach Fliesskomma
BDB3	48563	F99P9	Fliesskommakonstante 99999999.9
BDB8	48824	F9999	Fliesskommakonstante 999999999
BDBD	48573	F1E9	Fliesskommakonstante 1000000000
BDC2	48578	ERRZNR	Ausgabe der Zeilennummer bei Fehlermeldung
BDCD	48589	ADROUT	Zahl im Adreßbereich ausgeben; E: A/X enth. Hi/Low-Byte
BDDD	48605	FACASC	FAC nach ASCII-Format wandeln
BF71	49009	BBSQR	Basic-Befehl SQR
BF78	49016	FKHOCH	Potenzierung FAC = Konstante (A/Y) hoch FAC
BF7B	49019	FAHOCH	Potenzierung FAC = ARG hoch FAC
BFBF	49087		div. Fliesskommakonstanten für EXP
BFE8	49128	F1	Fliesskommakonstante 1
BFED	49133	BBEXP	Basic-Befehl EXP
E043	57411		Polynomberechnung
E059	57433		Polynomberechnung
E08D	57485		div. Fliesskommakonstanten für RND
E097	57495	BBRND	Basic-Befehl RND
E107	57607	OUTBREAK	Ausgabe von 'break'
E10C	57612	BSOUT	Basic-Aufruf der KERNAL-Routine BSOUT (mit Fehlerbehandl)

Adresse Hex.	Dez.	üblicher Name	Bemerkungen (A = Ausgabeparameter) (E = Eingabeparameter)
E112	57618	BASIN	BASIN ein Zeichen empfangen
E118	57624	CHKOUT	CKOUT Ausgabegeraet festsetzen
E11E	57630	CHKIN	CHKIN Eingabegeraet festsetzen
E124	57636	GETIN	GETIN ein Zeichen holen
E12A	57642	BBSYS	BASIC-Befehl SYS
E156	57686	BBSAVE	BASIC-Befehl SAVE
E165	57701	BBVERIF	BASIC-Befehl VERIFY
E168	57704	BBLOAD	BASIC-Befehl LOAD
E1BE	57790	BBOPEN	BASIC-Befehl OPEN
E1C7	57799	BBCLOSE	BASIC-Befehl CLOSE
E1D4	57812	PARLOSA	Parameter fuer LOAD und SAVE holen
E219	57881	PAROPEN	Parameter fuer OPEN holen
E264	57956	BBCOS	BASIC-Funktion COS
E26B	57963	BBSIN	BASIC-Funktion SIN
E2B4	58036	BBTAN	BASIC-Funktion TAN
E2E0	58080	FPI2	Fliesskommakonstante PI/2 = 1.57079633
E2E5	58085	F2PI	Fliesskommakonstante 2*PI = 6.28318531
E2EA	58090	F025	Fliesskommakonstante 0.25
E30E	58126	BBATN	BASIC-Funktion ATN
E37B	58235	BASNMI	BASIC-NMI-Einsprung
E394	58260	BASKALT	BASIC-Kaltstart
E3A2	58279	KCHRGET	Kopie der CHRGET-Routine
E3BA	58298		Anfangswert fuer RND-Funktion 0.811635157
E3BF	58303	BASICINI	BASIC-Initialisierungsroutine (setzt USR-Vektor/RAMTOP..)
E447	58439	BVECTAB	Tabelle der Standard BASIC-Vektoren
E453	58451	BVECLAD	Standard BASIC-Vektoren laden
E4E0	58592	WART	Wartet auf Commodore-Taste bzw. 8.5 Sekunden

Adresse Hex. Dez.	üblicher Name	Bemerkungen (A = Ausgabeparameter) (E = Eingabeparameter)
		Hier können Sie noch Routinen eintragen, die für Ihre Arbeiten wichtig sind !

Adresse Hex. Dez.	üblicher Name	Bemerkungen (A = Ausgabeparameter) (E = Eingabeparameter)

Adresse Hex. Dez.	üblicher Name	Bemerkungen (A = Ausgabeparameter) (E = Eingabeparameter)

KERNAL-Routinen

Die Betriebssystem-Routinen des KERNAL werden über eine standardisierte Sprungtabelle erreicht. Dort sind die Adressen von 39 Betriebssystem-Routinen festgelegt. Die Verwendung einer Sprungtabelle hat den Vorteil, daß Maschinenprogramme, die nur diese Sprungtabelle verwenden, nicht geändert werden brauchen, wenn Sie auf einen anderen Commodore-Rechner oder auf eine neue Version des Betriebssystems umgeschrieben werden sollen. Der Aufbau eines Betriebssystemaufrufs ist grundsätzlich folgender:

- Parameter bereitstellen
- Routine aufrufen
- Fehler behandeln
- Ergebnis auswerten.

Bei einer Ein-/Ausgabeaktion können Fehler auftreten, die durch ein gesetztes Carry-Flag nach der Rückkehr vom KERNAL-Unterprogramm gemeldet werden. Die Fehlernummer steht dann im Akkumulator. Folgende Fehlernummern sind möglich:

Fehlernummer	Bedeutung
0	Routine durch STOP-Taste unterbrochen
1	Zuviele Dateien sind offen
2	Die Datei ist bereits offen
3	Die Datei ist nicht offen
4	Datei nicht gefunden
5	Gerät nicht vorhanden
6	Die Datei ist keine Eingabe-Datei
7	Die Datei ist keine Ausgabe-Datei
8	Dateiname fehlt
9	Ungültige Geräteadresse
240	Die Speicherobergrenze wurde durch den RS-232 Puffer verändert

Es ist zu beachten, daß manche Ein- / Ausgabe-Routinen nicht diese Fehlermeldungen benützen, sondern man selbst durch den Aufruf der Routine READST den Eingabestatus feststellen muß.

Es folgt die Tabelle der KERNAL-Routinen, die ähnlich aufgebaut ist wie die Tabelle der Basic-Routinen.

Adresse Hex. Dez.	Name	Zweck	Ver. Reg.	Para meter	Bemerkungen
FFA5 65445	ACPTR	Hole 1 Byte v. seriellen Bus	A,X	a:A	Vorher TALK und evtl. TKSA anwenden; Fehler über READST
FFC6 65478	CHKIN	Öffne Kanal für Eingabe	A,X	e:X	Eing.par. ist log. Filenr.; vorher OPEN anwend.; beinhaltet TALK und TKSA
FFC9 65481	CHKOUT	Öffne Kanal für Ausgabe	A,X		Eing.par. ist log. Filenr.; vorher OPEN anwend.; beinhaltet TALK und TKSA
FFCF 65487	CHRIN	Hole ein Zeich. v. Eingabekanal	A,X	a:A	Für Eingabe nicht von Tastat. muß vorher CHKIN angew. werd. Bei Tastatur kann ganze Zeile geholt werden; Cursor blinkt.
FFD2 65490	CHROUT	Ein Zeichen auf Ausgabekanal	A	e:A	Für Ausgabe nicht auf BS muß vorher CHKOUT angew. werden
FFA8 65448	CIOUT	Ein Zeichen auf seriellen Bus		e:A	Vorher ist LISTEN und evtl. SECOND anzuwenden.
FF81 65409	CINT	Bildschirm und VIC initialis.	A,X,Y		Stellt Standardwerte im VIC her; löscht Bildschirm
FFE7 65511	CLALL	Schließt alle Dateien	A,X		Setzt Anz. off. Files auf 0; schließt Kanäle; E/A=Tast./BS
FFC3 65475	CLOSE	Schließt eine Datei	A,X,Y	e:A	Eingabepar. ist log. Filenr. im Akku; entspr. Basbef.CLOSE
FFCC 65484	CLRCHN	Schließt alle Kanäle	A,X		Schließt alle E/A-Kan.; wird die Routine nicht aufgerufen, kann es passieren, daß mehrere Geräte gleichz. aktiv sind
FFE4 65508	GETIN	Hole ein Zeich. aus Tast.puffer oder von RS-232	A,X,Y	a:A	Tastatur: Puffer leer -µ A=0 RS-232: Status muß mit READST gelesen werden

Adresse Hex.	Dez.	Name	Zweck	Ver. Reg.	Parameter	Bemerkungen
FFF3	65523	IOBASE	Definiert Page f.I/O-Chip CIA1	X,Y	a:X,Y	Liefert immer Adresse (X=Low, Y=High) des ersten I/O Chips
FF84	65412	IOINIT	Initialisiert I/O-Chips	A,X,Y		Setzt alle E/A-Chips und Routinen in definierten Zustand
FFB1	65457	LISTEN	Kommando an Ger am seriell. Bus	A	e:A	Eing.par. ist Geräteadr. 0-31 Gerät wartet auf Daten/READST
FFD5	65493	LOAD	Lade Daten von Eingabegerät (nicht Tastatur oder RS-232 in den Speicher	A,X,Y	e:A e:X,Y a:X,Y	Akku=0 f. Load; A=1 f. Verify Wenn Sekundäradresse(SA)=0, dann werden Daten ab der Adr. in X,Y gespeichert; wenn SA=1 dann Startadresse aus Header; X,Y enth. letzte gesp. Adr. Vorher SETLFS und SETNAM anw.
FF9C	65436	MEMBOT	Hole/Setze Beginn von verfügbaren RAM	X,Y	a:X,Y e:X,Y	Wenn Carry=1, dann Adr. lesen Wenn Carry=0, dann Startadresse RAM setzen
FFC0	65472	OPEN	Öffne logische Datei	A,X,Y		Vorher SETLFS und SETNAM anw. Fehler auch über READST lesen
FFF0	65520	PLOT	Hole/Setze Cursor-Position	A,X,Y	X,Y	X=Spalte;Y=Zeile; Wenn Carry=1,dann holen, C=0,dann setzen
FF87	65415	RAMTAS	RAM testen und MEMBOT/MEMTOP setzen	A,X,Y	e:A	Löscht Speicherbereich \$0000 bis \$0101 und \$0200 bis \$03FF setzt Beg. Video-Ram auf\$0400
FFDE	65502	RDTIM	Hole Zeit (1/60 sec.)	A,X,Y	a: A,X,Y	Holt Zeit; MSB im Akku; nächstes Byte im X-Reg. LSB in Y.
FFB7	65463	READST	Hole Statusbyte	A	a:A	

Adresse Hex. Dez.	Name	Zweck	Ver. Reg.	Para meter	Bemerkungen
					Raum für Notizen

Adresse Hex. Dez.	Name	Zweck	Ver. Reg.	Parameter	Bemerkungen
					Raum für Notizen

2.5 Laden der Maschinenprogramme

Ein kleines Maschinenprogramm kann natürlich mit POKE-Anweisungen direkt im Speicher abgelegt werden. Für größere Programme verwendet man meist einen Assembler, und dieser legt die Objekt-Datei in der Regel auf einer Diskette oder Kassette ab. Von dort kann man die Programme dann mit den folgenden Befehlen laden.

```
          LOAD "Name",8,1      (Floppy)

  oder    LOAD "Name",1,1      (Kassettenrekorder)
```

Anschließend muß meistens noch der Befehl NEW eingegeben werden, damit die Basic-Zeiger wieder zurückgesetzt werden, die durch ein LOAD im Direkt-Modus verändert worden sind.

Von einem Basic-Programm aus kann man die Maschinenprogramme ebenfalls laden, dabei muß man jedoch beachten, daß nach einem LOAD-Befehl das Programm wieder mit der ersten Programmzeile beginnt. Da die Variablen erhalten bleiben, kann man sich einen Merker setzen, der angibt, ob ein Maschinenprogramm bereits geladen worden ist. Das könnte wie folgt aussehen:

```
10 IF L=0 THEN L=1 : LOAD "NAME 1",8,1
20 IF L=1 THEN L=2 : LOAD "NAME 2",1,1
30 CLR : REM Variable L kann hier wieder entfernt werden
40 REM Beginn des Programms
```

Hier wird das erste Maschinenprogramm "Name 1" von Floppy geladen, und anschließend ein weiteres Programm "Name 2" vom Kassettenrekorder.

Wenn ein Maschinenprogramm zusammen mit dem Basic-Programm als ein Stück geladen werden soll, so muß das Maschinenprogramm im Anschluß an das Basic-Programm stehen. Das erreichen Sie beim Assembler durch entsprechende Wahl der Startadresse. Wenn sich Basic-Programm und Maschinenprogramm gleichzeitig im Speicher befinden, so können Sie durch Eingabe folgender Befehle den gesamten Speicherbereich als Programm abspeichern:

```
10 SYS 57812 "Name",8
20 POKE 193,(Anfangsadresse Low Byte)
30 POKE 194,(Anfangsadresse High Byte)
40 POKE 780,193
50 POKE 781,(Endadresse Low Byte)
60 POKE 782,(Endadresse High Byte)
70 SYS 65496
```

2.6 Anbinden mit dem SYS-Befehl

Die am häufigsten verwendete Vorgehensweise zum Anbinden von Maschinenprogrammen ist der SYS-Befehl. Trifft der Basic-Interpreter auf einen solchen Befehl, so verzweigt er zu einem Maschinenprogramm an der angegebenen Adresse. Dieses Maschinenprogramm muß mit RTS enden, womit die Programmkontrolle wieder an Basic übergeben wird. Umittelbar nach der Ausführung des SYS-Befehls steht der Basic-Programmzeiger auf dem nächsten Zeichen hinter der Adresse. Dadurch können mit entsprechenden Routinen leicht Parameter aus dem Basic-Text übernommen werden.

Parameterübergabe mit PEEK und POKE

Für den Anfänger ist es einfacher einige Parameter mit PEEK- und POKE-Befehlen zu übernehmen bzw. zu übergeben. Die Übernahmeadressen müssen ebenso wie das Maschinenprogramm selbst, in einem Bereich liegen, der von Basic nicht verändert wird (z.B. von $C000 bis $CFFF).

Einige der für die Parameterübergabe verwendbaren Speicherzellen sind bereits vom Betriebssystem her vorgegeben. Das sind die Speicherzellen 780 bis 783 (dezimal). Vor dem Aufruf einer Maschinen-Routine mit SYS kann man hier nacheinander die Werte für Akkumulator, X-Register, Y-Register und Prozessorstatus mit POKE-Befehlen festlegen, die dann vor dem eigentlichen Abarbeiten der Maschinen-Routine so besetzt werden. Nach Beendigung der Maschinen-Routine mit RTS werden die eventuell veränderten Werte dieser Register in diese Speicherzellen zurückgeschrieben, und können dann mit PEEK-Befehlen ausgelesen werden. Diese Tatsache bietet Ihnen die Möglichkeit auch Betriebssystem-Routinen vom Basic aus aufzurufen, die einen bestimmten Wert im Akku oder den anderen Registern benötigen. Ein Beispiel dafür ist das in Kapitel 2.5 angesprochene Speichern eines zusammenhängenden Speicherbereichs mit Hilfe der KERNAL-Routine SAVE.

Parameterübergabe beim erweiterten SYS-Befehl

Ein SYS-Befehl mit Parametern kann z.B folgende Form haben:

SYS Adresse,Param1,Param2,Param3

Die Trennung der Paramter voneinander und von der Adresse geschieht hier mit Kommata. Dies ist nicht unbedingt notwendig, jedoch ist diese Art am leichtesten vom Maschinenprogramm her zu bearbeiten. Wie oben erwähnt, steht nach dem Aufruf der Routine der Basic-Programm-Zeiger unmittelbar hinter der Aufrufadresse, d.h. in unserem Beispiel auf dem Komma. Es muß jetzt geprüft werden, ob nach der Adresse des SYS-Befehles wirklich ein Komma folgt. Das kann mit dem Unterprogrammaufruf JSR CHKCOM (vgl. Basic-Routinen) durchgeführt werden. Diese Routine setzt auch gleich den Basic-Programmzeiger eine Stelle weiter. Dann kann man den folgenden Parameter mit einer entsprechenden Betriebssystem-Routine auswerten, dazu sind vor allem folgende Basic-Routinen geeignet:

FRMNUM holt numerischen Ausdruck in Fließkomma-Akkumulator
GETBYT holt Byte-Wert in X-Register
GETAB holt Adresswert nach $14,$15 und Bytewert ins X-Register
VARSUC bestimmt Adresse einer Variablen
FRMEVL wertet beliebigen Ausdruck aus; bei String-Parametern ist zusätzlich FRESTR aufzurufen

Die Beschreibung der Routinen finden Sie bei den Basic-Routinen, ausführliche Beispiele in Kapitel 4.

2.7 USR-Funktion

Trifft der Basic-Interpreter beim Auswerten eines Ausdrucks auf die Funktion USR, so wertet er automatisch den Parameter dieser Funktion (den in Klammern angegebenen Wert) aus, und ruft dann eine Maschinen-Routine auf, deren Adresse in den Speicherzellen 785 und 786 für das Low- und das High-Byte gespeichert ist.

Wird die USR-Funktion als numerische Funktion eingesetzt, wie das meist der Fall ist, so ist das Ergebnis der Funktion mit dem Inhalt des Fließkomma-Akkumulators identisch. Den Fließkomma-Akkumulator (FAC) kann man mit entsprechenden ROM-Routinen (vergl. Kapitel 2.4) verändern.

Prinzipiell ist es auch möglich die USR-Funktion als

String-Funktion zu benützen, doch wollen wir darauf nicht weiter eingehen.

Die Übergabe von Parametern an die USR-Funktion kann sehr mannigfaltig gestaltet werden. Im einfachsten Fall wird ein numerisches Argument übergeben, das der Maschinen-Routine vom Basic-Interpreter im Fließkomma-Akkumulator zur Verfügung gestellt wird.

Es kann aber auch ein String-Parameter übergeben werden. Die Werte dieses Parameters müssen dann durch Aufruf der Routine FRESTR bestimmt werden. Ein Beispiel dafür finden Sie in Kapitel 4.8, in dem der Wert einer hexadezimalen Zahl bestimmt wird.

Auch können noch mehr Parameter an die USR-Funktion übergeben werden. Der Basic-Interpreter stellt den ersten Parameter bereits aufbereitet zur Verfügung. Die anderen Parameter müssen wie beim SYS-Befehl gelesen werden.

Ein wesentlicher Nachteil der USR-Funktion ist, daß nur ein Maschinenprogramm damit aufgerufen werden kann. Wenn man aber als zweiten Parameter die Nummer eines Programms übergibt, so kann man damit einen Sprungverteiler aufrufen, der dann entsprechend zu den einzelnen Programmen verzweigt. Man kann natürlich auch immer wieder den USR-Vektor verändern.

2.8 Die Zero-Page des Commodore 64

Der wohl wichtigste Bereich eines 6510-Rechners ist der Speicherbereich von 0 bis 255, die sogenannte Zero-Page. Im Anschluß an diese einleitenden Worte finden Sie eine Tabelle mit den Belegungen, wie sie durch das Commodore-Basic und das Betriebssystem benützt werden. Die Adressen sind sowohl hexadezimal als auch dezimal angegeben. In der Spalte 'Bedeutung' finden Sie die verwendete Belegung dieser Speicherzellen. Die Basic-Routinen verwenden diese Speicherzellen manchmal auch zu anderen Zwecken, es sind jeweils aber nur die wichtigsten aufgeführt.

Diese Tabelle ist aus zwei Gründen wichtig. Zum einen können Sie nachsehen, wo sich ein von Ihnen gewünschter Zeiger befindet, andererseits enthält diese Tabelle auch Hinweise darüber, welche Speicherzellen Sie für Ihre eigenen Zwecke verwenden können, ohne daß Sie von Basic verändert werden.

Es gibt in der gesamten Zero-Page fast keine Speicher-

zelle, die nicht irgendwann von Basic oder KERNAL verändert wird. Sie können aber, weil Sie Ihr Programm ja kennen, entscheiden, ob eine bestimmte Funktion benötigt wird oder nicht. Z.B. sind die Fließkomma-Akkumulatoren #3 und #4 verwendbar, wenn keine komplizierten arithmetischen Ausdrücke vorkommen. Ebenso können Sie Adressen verwenden, die sich auf den Kassettenrekorder oder die RS-232-Schnittstelle beziehen, wenn Sie diese nicht verwenden.

Hexadresse	**Dezimal**	**Bedeutung**
00	0	Datenrichtungsregister für Prozessor-Port
01	1	Prozessor-Port
02	2	unbenutzt
03 - 04	3 - 4	Vektor für Umwandlung Fließkomma nach Integer
05 - 06	5 - 6	Vektor für Umwandlung Integer nach Fließkomma
07	7	Suchzeichen
08	8	"-Merker
09	9	Speicher für Spalte des TAB-Befehls
0A	10	Load = 0,Verify = 1
0B	11	Zeiger in Eingabepuffer, Anzahl der Dimensionen
0C	12	Merker für automatische Dimensionierung
0D	13	Datentyp: $00=numerisch, $FF=String
0E	14	Datentyp: $00=real, $80=integer
0F	15	"-Merker bei List
10	16	Merker für FN
11	17	Merker für Input=$00,GET=$40,READ=$98
12	18	Vorzeichen bei ATN
13	19	aktive I/O-Datei
14 - 15	20 - 21	Adresswert, z.B. Zeilennummer
16	22	Zeiger auf Stringstack
17 - 18	23 - 24	Zeiger auf zuletzt verwendeten String
19 - 21	25 - 33	Stringstack
22 - 25	34 - 37	Zeiger für diverse Zwecke
26 - 2A	38 - 42	Register für Funktionsauswertung und Arithmetik
2B - 2C	43 - 44	Zeiger auf Basic-Programmstart
2D - 2E	45 - 46	Zeiger auf Start der Variablen
2F - 30	47 - 48	Zeiger auf Start der Arrays
31 - 32	49 - 50	Zeiger auf Ende der Arrays
33 - 34	51 - 52	Zeiger auf Beginn der Strings
35 - 36	53 - 54	Hilfszeiger für Strings
37 - 38	55 - 56	Zeiger auf Basic-RAM Ende
39 - 3A	57 - 58	augenblickliche Basic-Zeilennummer
3B - 3C	59 - 60	vorherige Basic-Zeilennummer

Hexadresse	Dezimal	Bedeutung
3D - 3E	61 - 62	Zeiger auf nächstes Basic-Statement für CONT
3F - 40	63 - 64	augenblickliche Zeilennummer für DATA
41 - 42	65 - 66	Zeiger auf DATA-Element
43 - 44	67 - 68	Vektor für INPUT-Routine
45 - 46	69 - 70	Variablenname
47 - 48	71 - 72	Variablenadresse
49 - 4A	73 - 74	Variablenzeiger für FOR...NEXT
4B - 4C	75 - 76	Zwischenspeicher für Programmzeiger
4D	78 - 79	Zeiger für FN
50 - 53	80 - 83	Zeiger für diverse Zwecke
54	84	Konstante $4C JMP für Funktionen
55 - 56	85 - 86	Sprungvektor für Funktionen
57 - 5B	87 - 91	Fließkomma-Akku #3
5C - 60	92 - 96	Fließkomma-Akku #4
61 - 65	97 -101	Fließkomma-Akku #1, FAC
66	102	Vorzeichen des FAC
67	103	Zähler für Polynom-Auswertung
68	104	Überlaufbyte des FAC
69 - 6D	105-109	Fließkomma-Akku #2, ARG (Argument)
6F	111	Vergleichsbyte der Vorzeichen von FAC und ARG
70	112	Rundungsbyte für FAC
71 - 72	113-114	Zeiger für Kassettenpuffer
73 - 8A	115-138	CHRGET - Routine, holt Zeichen aus Basic-Text
7A - 7B	122-123	Programmzeiger (Basic)
8B - 8F	139-143	letzter RND-Wert
90	144	Statuswort ST
91	145	Merker für Stop-Taste
92	146	Zeitkonstante für Kassettenrekorder
93	147	Merker für Load $00 oder Verify $01
94	148	Merker für Ausgabe (serieller Bus)
95	149	Ausgabepuffer für seriellen Bus
96	150	Merker für EOT vom Kassettenrekorder
97	151	Hilfszelle (Zwischenspeicher)
98	152	Anzahl der offenen Files
99	153	aktives Eingabegerät
9A	154	aktives Ausgabegerät
9B	155	Parität für Kassettenrekorder
9C	156	Merker für Byte empfangen
9D	157	Merker für Direkt-Modus $80 oder Programm $00
9E	158	Prüfsumme erster Durchlauf beim Einlesen von Daten von Kassettenrekorder
9F	159	Prüfsumme zweiter Durchlauf
A0 - A2	160-162	Uhr (1/60 Sek.)
A3	163	Bitzähler für serielle Ausgabe
A4	164	Zähler für Kassettenrekorder

Hexadresse	Dezimal	Bedeutung
A5	165	Zähler für Kassettenrekorder-Synchronisation
A6	166	Zeiger für Kassettenrekorder-Puffer
A7 - AB	167-171	Arbeitsspeicher für Ein-/Ausgabe der RS 232-Schnittstelle
AC - AD	172-173	Zeiger für Kassettenpuffer und Scrolling
AE - AF	174-175	Zeiger auf Programmende bei LOAD/SAVE
B0 - B1	176-177	Zeitkonstanten für Kassettenrekorder-Timing
B2 - B3	178-179	Zeiger auf Kassettenpuffer
B4	180	Bitzähler für RS 232
B5	181	nächstes Bit für RS 232
B6	182	Puffer für auszugebendes Byte
B7	183	Länge des aktuellen Dateinamens
B8	184	aktuelle logische Dateinummer
B9	185	aktuelle Sekundäradresse
BA	186	aktuelle Gerätenummer
BB - BC	187-188	Zeiger auf aktuellen Dateinamen
BD	189	Arbeitsspeicher serielle Ein/Ausgabe
BE	190	Blockzähler für Kassettenrekorder
BF	191	Puffer für serielle Ausgabe
C0	192	Merker für Kassettenmotor
C1 - C2	193-194	Startadresse für Ein/Ausgabe
C3 - C4	196-196	Endadresse für Ein/Ausgabe
C5	197	Letzte gedrückte Taste (64=keine Taste gedrückt)
C6	198	Anzahl der gültigen Zeichen im Tastaturpuffer
C7	199	Merker für Reverse-Modus
C8	200	Zeilenende für Eingabe
C9	201	Cursorzeile für Eingabe
CA	202	Cursorspalte für Eingabe
CB	203	gedrückte Taste (keine Taste: 64)
CC	204	Merker für blinkenden Cursor (0:blinken)
CD	205	Zähler für Cursor blinken
CE	206	Zeichen unter dem Cursor
CF	207	Merker für Cursor-Blinken
D0	208	Merker für Eingabe von Tastatur oder Bildschirm
D1 - D2	209-210	Zeiger auf Start der aktuellen Bildschirmzeile
D3	211	Cursorspalte
D4	212	"-Merker für Editor
D5	213	Länge der Bildschirmzeile
D6	214	Cursorzeile
D7	215	diverse Zwecke
D8	216	Anzahl der Inserts

Hexadresse	Dezimal	Bedeutung
D9 - F2	217-242	MSB der Bildschirmzeilenanfänge
F3 - F4	243-244	Zeiger in Farb-RAM
F5 - F6	245-246	Zeiger auf Tastatur-Dekodiertabelle
F7 - F8	247-248	Zeiger auf RS 232 Eingabepuffer
F9 - FA	249-250	Zeiger auf RS 232 Ausgabepuffer
FB - FE	251-254	**Frei für Benutzer**
FF	255	Eingabepuffer für Basic

3

Basic-Programm des Assemblers

3. Basic-Programm des Assemblers

Das folgende Kapitel ähnelt sehr stark dem in Band 1 vorgestellten Assembler. Anders als bei dem Assembler aus Band 1 werden hier später einige Unterprogramme selbst in Assemblersprache geschrieben, die wir deshalb hier im Basic-Teil auch nicht ausführlich besprechen werden.

Man braucht, wie wir bereits im ersten Kapitel gesehen haben, zur Entwicklung von Maschinenprogrammen ein eigenes Programm, das die Zeichenreihen, die einen Befehl darstellen sollen, in die für den Prozessor lesbaren Werte umwandelt. Ein solches Programm nennt man Assembler. Wenn wir dabei noch Symbole für konstante Werte und Marken für Sprungziele Marken verwenden, so haben wir einen symbolischen Assembler, d.h. wir können jetzt schreiben: 'JMP ENDE' anstatt z.B. 76, 255, 192.

Ein Assembler muß aber noch mehr leisten. Er muß den Text, der in Maschinensprache übersetzt werden soll, von einem externen Speichermedium lesen können, er muß die Symbole verwalten und er muß das Resultat wieder auf ein externes Speichermedium zurückschreiben. Mit all diesen Fähigkeiten ist der im vorliegenden Band beschriebene Assembler ausgestattet. Zusätzlich kann man dann noch den Assemblierungsvorgang mit Direktiven steuern, so z.B. die Wahl der Startadresse oder das Hinzufügen von bereits früher erstellten Programmen.

Zur Erläuterung der Funktionsweise eines Assemblers sind alle Unterprogramme in Basic abgebildet. Wir erläutern bei jedem Unterkapitel, welche Änderungen im Hauptprogramm dazu notwendig sind. Das komplette Basic-Listing der 'gemischten Version' ist im Anhang abgedruckt.

Wir haben vier Unterkapitel vorgesehen. Zunächst gehen wir auf die DATA-Statements ein, die unter anderem die verschiedenen mnemotechnischen Bezeichnungen, wie Sie bei dem 6510/6502 üblich sind, inklusive ihrer Parameter angeben. Als nächstes werden die Basic-Unterprogramme erläutert, welche später durch Maschinenroutinen ersetzt werden.

Im dritten Teil wird auf die anderen Unterprogramme eingegangen, und im vierten Unterkapitel wird schließlich das Hauptprogramm beschrieben.

3.1 DATA-Anweisungen

Die DATA-Statements liegen im Basic-Programm ab Zeile 54000 bis hin zu Zeile 58260. Die DATA-Statements selbst sind wieder wie folgt untergliedert:

54000 Anzahl der Adressenmodes je Befehlstyp
55000 Angabe der mnemotechnischen Befehle mit ihrem dreibuchstabigen Code sowie die Nummer des Typs und alle aufgrund des Typs möglichen späteren Maschinencodes
56000 Direktiven; Anweisungen im Assemblerprogramm, die keinen Maschinencode erzeugen
57000 Basic-Schlüsselwörter
58000 Fehlermeldungen

3.1.1 Adressierungsarten

```
54000 REM %  AC(0..5)      ANZAHL MODES                    %
54010 DATA1:REM TYP 0 IMPLIED
54020 DATA2:REM TYP 1 (JUMPS) ABSOLUTE,INDIRECT
54030 DATA10:REM #,Z,ZX,ZY,A,AX,AY,IX,IY,AC  TYP 2
54040 DATA0:REM BYTE
54050 DATA0:REM WORD
54060 DATA1:REM TYP 5 RELATIVE BRANCHES
```

Die Adressierungsarten werden später in dem Feld AC(0..5) gespeichert. Folgende Anzahlen für Adressenmodes sind möglich:

Typ 0 - Modes: 1,d.h. ein dreibuchstabiger Assemblerbefehl des Typ 0, kann nur in einen einzigen Maschinenbefehl umgewandelt werden.

Typ 1 - Modes: 2, hauptsächlich Jump-Befehle sind vom Typ 1. Diese können absolut und indirekt adressiert werden

Typ 2 - bis 10 Modes, in Typ 2 sind alle möglichen Operationen zusammengefaßt, die in bis zu zehn verschiedene Maschinencodes umgewandelt werden können.

Typ 3 - Modes: 0, erzeugt keinen Code, sondern ist

ein Pseudobefehl zum Reservieren einer Konstanten von einem Byte

Typ 4 - Modes: 0, erzeugt ebenfalls keinen Code, sondern dient zum Reservieren einer Konstanten von zwei Byte (= 1 Word)

Typ 5 - Modes: 1, faßt alle Befehle für relative Sprünge zusammen.

3.1.2 Mnemotechnische Befehle

```
55000 REM K$(1..58),KM(1..58),KC$(1..58,0..AC(KM(I))-1)
55040 DATAADC,2,69,65,75,,6D,7D,79,61,71,
55050 DATAAND,2,29,25,35,,2D,3D,39,21,31,
55060 DATAASL,2,,06,16,,0E,1E,,,,0A
55070 DATABCC,5,90
55080 DATABCS,5,B0
55090 DATABEQ,5,F0
55100 DATABIT,2,,24,,,2C,,,,,
55110 DATABMI,5,30
55120 DATABNE,5,D0
55130 DATABPL,5,10
55140 DATABRK,0,00
55150 DATABVC,5,50
55160 DATABVS,5,70
55170 DATABYT,3
55180 DATACLC,0,18
55190 DATACLD,0,D8
55200 DATACLI,0,58
55210 DATACLV,0,B8
55220 DATACMP,2,C9,C5,D5,,CD,DD,D9,C1,D1,
55230 DATACPX,2,E0,E4,,,EC,,,,,
55240 DATACPY,2,C0,C4,,,CC,,,,,
55250 DATADEC,2,,C6,D6,,CE,DE,,,,
55260 DATADEX,0,CA
55270 DATADEY,0,88
55280 DATAEOR,2,49,45,55,,4D,5D,59,41,51,
55290 DATAINC,2,,E6,F6,,EE,FE,,,,
55300 DATAINX,0,E8
55310 DATAINY,0,C8
55320 DATAJMP,1,4C,6C
55330 DATAJSR,1,20,
55340 DATALDA,2,A9,A5,B5,,AD,BD,B9,A1,B1,
55350 DATALDX,2,A2,A6,,B6,AE,,BE,,,
55360 DATALDY,2,A0,A4,B4,,AC,BC,,,,
55370 DATALSR,2,,46,56,,4E,5E,,,,4A
55380 DATANOP,0,EA
55390 DATAORA,2,09,05,15,,0D,1D,19,01,11,
55400 DATAPHA,0,48
```

```
55410 DATAPHP,0,08
55420 DATAPLA,0,68
55430 DATAPLP,0,28
55440 DATAROL,2,,26,36,,2E,3E,,,,2A
55450 DATAROR,2,,66,76,,6E,7E,,,,6A
55460 DATARTI,0,40
55470 DATARTS,0,60
55480 DATASBC,2,E9,E5,F5,,ED,FD,F9,E1,F1,
55490 DATASEC,0,38
55500 DATASED,0,F8
55510 DATASEI,0,78
55520 DATASTA,2,,85,95,,8D,9D,99,81,91,
55530 DATASTX,2,,86,,96,8E,,,,,
55540 DATASTY,2,,84,94,,8C,,,,,
55550 DATATAX,0,AA
55560 DATATAY,0,A8
55570 DATATYA,0,98
55580 DATATSX,0,BA
55590 DATATXA,0,8A
55600 DATATXS,0,9A
55610 DATAWOR,4
```

In diesen DATA-Statements sind alle mnemotechnischen Befehle des 6510/6502 zusammengefaßt. Jede DATA-Zeile besteht aus dem dreibuchstabigen Befehl, dem Typ, wie er in Kapitel 5.1.1 beschrieben wurde, und den verschiedenen hexadezimalen Werten, die das Maschinenprogramm aufgrund des Codes und den verschiedenen Adressierungen annehmen kann.

Bei Verwendung des später erläuterten Maschinenprogramms zum Suchen einer mnemotechnischen Bezeichnung werden die Befehle selbst in dieser Tabelle überflüssig. Die Maschinenroutine beinhaltet diese Tabelle selbst.

3.1.3 Direktiven

```
56030 DATAORG,END,INO,DUP,ASC,INCLUDE
```

Der vorliegende Assembler kennt sechs Direktiven:

%ORG	- erstes absolutes Byte (Startadresse) für das Maschinenprogramm (Objekt-Programm) festlegen
%END	- Ende des Quellprogramms
%INO	- Startadresse für das Assembler-Programm am Bildschirm erfragen
%DUP	- Dupliziere Befehl um angegebene Anzahl
%ASC	- Speichere Text an dieser Stelle in Objekt-Datei (als Folge von ASCII-Codes)
%INCLUDE	- Füge an dieser Stelle eine andere Quell- - Datei ein

In den DATA-Statements werden die Direktiven ohne '%' angegeben.

3.1.4 Basic-Keywords

```
57010 REM %  BK$(128..203)      BASIC-KEYWORDS                    %
57030 DATAEND,FOR,NEXT,DATA,INPUT#,INPUT,DIM
57040 DATAREAD,LET,GOTO,RUN,IF,RESTORE
57050 DATAGOSUB,RETURN,REM,STOP,ON,WAIT,LOAD
57060 DATASAVE,VERIFY,DEF,POKE,PRINT#
57070 DATAPRINT,CONT,LIST,CLR,CMD,SYS,OPEN
57080 DATACLOSE,GET,NEW,TAB(,TO,FN,SPC(
57090 DATATHEN,NOT,STEP,+,-,*,/,↑,AND,OR,>,=
57100 DATA<,SGN,INT,ABS,USR,FRE,POS,SQR
57110 DATARND,LOG,EXP,COS,SIN,TAN,ATN,PEEK
57120 DATALEN,STR$,VAL,ASC,CHR$,LEFT$
57130 DATARIGHT$,MID$,GO
```

In einer Programmdatei sind die Basic-Keywords jeweils in einem Byte dargestellt, deren Codes etwas abseits von den normalen Buchstaben im ASCII-Bereich von 128 bis 203 liegen. Deshalb wird in dem Feld BK$() dieser Bereich mit den Basic-Keywords besetzt. Die Bedeutung der Basic-Keywords dürfte Ihnen bekannt sein. Notwendig ist die Verarbeitung der Basic-Keywords, da in den Assembler-Programmen eventuell Basic-Keywords vorkommen können, die da nichts zu suchen haben.

Das Umsetzen einer Programmzeile in Klartext erfolgt später in einem Maschinenprogramm, welches ähnlich wie die LIST-Routine arbeitet. Die oben abgebildeten DATA-Statements sind dann nicht mehr notwendig.

3.1.5 Fehlermeldungen

```
58010 REM %  ER$(1..E9)      FEHLERMELDUNGEN                    %
58030 DATA"SYNTAX : FALSCHES MNEMONIC"
58040 DATA"SYNTAX : FALSCHE DIRECTIVE"
58050 DATA"SYNTAX : OPERAND FEHLT"
58060 DATA"SYNTAX : KLAMMERN FALSCH GESETZT"
58070 DATA"SYNTAX : MODEZEICHEN NICHT AN ERSTER STELLE"
58080 DATA"ADRESSIERUNGSART HIER NICHT MOEGLICH"
58090 DATA"KEIN OPERAND ERLAUBT"
58100 DATA"OPERAND MUSS EINE KONSTANTE SEIN"
58110 DATA"OPERAND MUSS EINE MARKE (LABEL) SEIN"
58120 DATA"OPERAND FALSCH SPEZIFIZIERT"
```

```
58130 DATA"MARKE ODER KONSTANTE SCHON DEFINIERT"
58140 DATA"AUSDRUCK DARF NUR 2 ARGUMENTE ENTHALTEN"
58150 DATA"WERT ZU GROSS"
58160 DATA"FALSCHE LAENGE EINER HEX- ODER BINAER-ZAHL"
58170 DATA"UNGUELTIGES ZEICHEN IN BINAERZAHL"
58180 DATA"DIESER WERT MUSS BEREITS HIER DEFINIERT SEIN"
58190 DATA"SPRUNG ZU WEIT"
58200 DATA"ZU VIELE FEHLER"
58210 DATA"STARTADRESSE NICHT DEFINIERT"
58220 DATA"STARTADRESSE BEREITS DEFINIERT"
58230 DATA"UNGUELTIGE HEXADEZIMALZAHL"
58240 DATA"ZU VIELE SYMBOLE"
58250 DATA"DATEI NICHT GEFUNDEN"
58260 DATA"UNGUELTIGER DATEINAME"
```

3.2 Unterprogramme

In diesem Kapitel werden die einzelnen Unterprogramme vorgestellt, wobei die im ersten Teil angeführten Unterprogramme später assembliert werden. Die anderen Unterprogramme werden seltener gebraucht, und ihre Assemblierung würde keinen großen Zeitvorteil im Assemblerlauf bringen. Außerdem sollten diese Unterprogramme für Ihre Zwecke änderbar sein, was in Basic wesentlich einfacher ist.

3.2.1 Häufiger verwendete Unterprogramme

Im folgenden sind die Unterprogramme abgebildet, die später durch Maschinenprogramme ersetzt werden. Die Funktionsweise dieser Routinen braucht hier nicht weiter erläutert werden, da es sich um recht einfache Algorithmen handelt. Im Zweifelsfall sei auf Band 1 verwiesen.

Als Parameter werden Variablen und Speicherzellen verwendet. Die Auswahl der Parameter mag manchmal etwas umständlich erscheinen, weil die Struktur der Untergprogrammaufrufe für den späteren Einsatz von Maschinenroutinen vorbereitet wurde.

Folgende Routinen sind abgebildet:

10	Ein Zeichen lesen aus File #FP
50	Position (A) von A$ in AA$ bestimmen
100	Blanks von T$ eliminieren
200	2-stellige Hexzahl (H$) aus H bilden
250	4-stellige Hexzahl (HH$) aus HH bilden
400	Wert von Hexzahl (HH$) nach HH
500	Name in Tabelle suchen und Werte bestimmen
600	Check auf mnemotechnischen Code

600	Check auf mnemotechnischen Code
700	Ein Sonderzeichen (A$) in T$ suchen
800	Mehrere Sonderzeichen (aus AA$) in T$ suchen
17000	Fehler registrieren
18000	Symbol in Tabelle einfügen
20500	Symboltabelle drucken
20700	Symboltabelle speichern
22500	Eine Programmzeile lesen
22700	Eine Textzeile lesen
50500	Initialisierungsroutine (Teil des Vorspanns)

In der ersten Zeile jedes Unterprogramms ist der SYS-Befehl angegeben, mit dem später der Aufruf des Unterprogramms ersetzt wird.

Zeichen aus Datei einlesen

```
10 REM *** ENTSPRICHT SYS P2,FP ***
15 GET#FP,I$
20 IS=ST
25 IFI$=""THENI$=CHR$(0)
30 POKEQ3,ASC(I$)
35 RETURN
```

Eingabeparameter: FP = Nummer der Eingabedatei
Ausgabeparameter: Zelle Q3 = ASCII-Code des eigentlichen Zeichens
IS = Eingabestatus

Indexfunktion

```
50 REM *** ENTSPRICHT  SYS P8,A$,T$,A  ***
55 A=0
60 AA=LEN(A$)
65 IFAA>LEN(T$)THEN85
70 FORA=1TOLEN(T$)-AA+1
75 IFMID$(T$,A,AA)<>A$THENNEXT
80 IFA>LEN(T$)-AA+1THENA=0
85 RETURN
```

Eingabeparameter: A$ = Zeichenreihe, die gesucht wird
AA$ = Zeichenreihe, in der gesucht wird
Ausgabeparameter: A = Position von A$ in AA$

Leerzeichen eliminieren

```
100 REM *** ENTSPRICHT SYS P1,T$ ***
110 IFRIGHT$(T$,1)=" "THENT$=LEFT$(T$,LEN(T$)-1):GOTO110
120 IFLEFT$(T$,1)=" "THENT$=MID$(T$,2):GOTO120
130 RETURN
```

Ein-/Ausgabeparameter: T$ = Zeichenreihe, deren Leerzeichen eliminiert werden sollen

Zweistellige Hexadezimalzahl bilden

```
200 REM *** ENTSPRICHT SYS P6,H,H$  ***
210 H$=MID$(HE$,H/16+1,1)+MID$(HE$,(HAND15)+1,1)
220 RETURN
```

Eingabeparameter: H = Zahlwert
Ausgabeparameter: H$ = Hexadezimalzahl (2 Zeichen)
Bemerkung: HE$ = "0123456789ABCDEF"

Vierstellige Hexadezimalzahl bilden

```
250 REM *** ENTSPRICHT SYS P5,HH,HH$ ***
260 H=INT(HH/256)
270 GOSUB200
280 HH$=H$
290 H=HH-256*H
300 GOSUB200
310 HH$=HH$+H$
320 RETURN
```

Eingabeparameter: H = Zahlwert
Ausgabeparameter: H$ = Hexadezimalzahl (2 Zeichen)

Wert von Hexadezimalzahl bestimmen

```
400 REM *** ENTSPRICHT  HH=USR(HH$)   ***
410 HH=0
420 FORH=1TOLEN(HH$)
430 HH=16*HH+FNV(ASC(MID$(HH$,H,1)))
440 NEXT
450 RETURN
```

Eingabeparameter: HH$ = Hexadezimalzahl (bis 4 Zeichen)
Ausgabeparameter: HH = Zahlwert
Bemerkung: FNV(X) = X-48+7*(X größer als 64)

Tabelle durchsuchen und Wert bestimmen

```
500 REM *** ENTSPRICHT SYS PB,TN$,TY,TV  ***
510 TY=0
520 TV=0
530 IFTA=0THENRETURN
540 FORTF=1TOTA
550 IFTN$(TF)<>TN$THENNEXT
560 IFTF>TATHENTY=0:TV=0:RETURN
570 TV=TV(TF)
580 TY=TY(TF)
590 RETURN
```

Eingabeparameter: TN$ = zu suchendes Symbol
Ausgabeparameter: TF = Nummer des Symbols
TV = Wert des Symbols
TY = Typ des Symbols

Test auf Vorhandensein der mnemotechnischen Bezeichnung

```
600 REM *** ENTSPRICHT SYS PD,A$ :K=PEEK(782) ***
610 FORK=1TO58
620 IFA$<>K$(K)THENNEXT
630 IFK>58THENK=0
650 RETURN
```

Eingabeparameter: K$ = zu suchende Bezeichnung
Ausgabeparameter: K = Nummer des Bezeichnung

Ein Sonderzeichen in T$ suchen

```
700 REM *** ENTSPRICHT ETWA SYSP9,T$,A,A$ ***
710 A=0
720 IFT$=""THENRETURN
730 FORA=1TOLEN(T$)
740 IFMID$(T$,A,1)<>A$THENNEXT
750 IFA>LEN(T$)THENA=0
760 RETURN
```

Eingabeparameter: A$ = zu suchendes Zeichen
T$ = Text, in dem gesucht wird
Ausgabeparameter: A = Position des Zeichens

Mehrere Zeichen (aus AA$) in T$ suchen

```
800 REM *** ENTSPRICHT ETWA SYSP9,T$,A(0),AA$ ***
810 FORAA=1TOLEN(AA$)
820 A$=MID$(AA$,AA,1):GOSUB700:A(AA-1)=A
830 NEXTAA
840 RETURN
```

```
Eingabeparameter:  AA$ = alle zu suchenden Zeichen
                   T$  = Text, in dem gesucht wird
Ausgabeparameter:  A() = Positionen der Zeichen
```

Fehler registrieren

```
17000 REM *** ENTSPRICHT  SYS PA,ER   ***
17010 EB=PEEK(Q4)
17020 EB=EB+1
17030 IFEB>=15THENEB=15:ER=18
17040 POKEQ4+EB,ER
17050 POKEQ4,EB
17060 RETURN
```

```
Eingabeparameter      : ER        = Fehlernummer
Ein-/Ausgabeparameter: Zelle Q4  = Anzahl Fehler
```

Symbol in Tabelle einfügen

```
18000 REM *** ENTSPRICHT  SYS PC,TN$,TV,TY   ***
18005 H1=TV:H2=TY
18010 GOSUB500
18020 IFTYTHENER=11:GOSUB17000:RETURN
18025 TV=H1:TY=H2
18030 TA=TA+1
18040 TY(TA)=TY
18050 TV(TA)=TV
18060 TN$(TA)=TN$
18070 RETURN
```

```
Eingabeparameter      : TN$  = Name des Symbols
                        TV   = Wert des Symbols
                        TY   = Typ  des Symbols
Ein-/Ausgabeparameter: TA   = Anzahl Symbole
Bemerkung             : Die Werte TV und TY müssen in H1
```

und H2 zwischengespeichert werden, weil das Unterprogramm ab Zeile 500 die Variablen TV und TY auch dann verändert, wenn das Symbol nicht gefunden wurde.

Symboltabelle drucken

```
20500 REM *** ENTSPRICHT  SYS PE  ***
20510 IFTA=0THENRETURN
20520 FORI=1TOTA
20530 PRINTTN$(I);" ";MID$("UCL",TY(I)+1,1);" ";
20540 HH=TV(I):GOSUB250
20550 PRINTHH$
20560 NEXTI
20570 RETURN
```

Symboltabelle speichern

```
20700 REM *** ENTSPRICHT  SYS PF,5  ***
20710 IFTA=0THENRETURN
20720 FORI=1TOTA
20730 PRINT#5,MID$("UCL",TY(I)+1,1);",";TN$(I);",";TV(I)
20740 NEXTI
20750 RETURN
```

Eine Programmzeile lesen

```
22500 REM *** ENTSPRICHT SYS P3,FP,T$ ***
22510 T$=""
22520 GOSUB10
22530 GOSUB10
22540 POKEQ2+1,PEEK(Q3)
22550 IFPEEK(Q3)=0THENRETURN
22560 GOSUB10
22570 POKEQ1,PEEK(Q3)
22580 GOSUB10
22590 POKEQ1+1,PEEK(Q3)
22600 GOSUB10
22610 IFPEEK(Q3)=0THEN22640
22620 T$=T$+BK$(PEEK(Q3))
22630 GOTO22600
22640 RETURN
```

```
Eingabeparameter: FP           = Logische Dateinummer der
                                 Eingabedatei
Ausgabeparameter: T$           = eingelesene Textzeile
                  Zelle Q1     = Zeilennummer  Low-Byte
                  Zelle Q1+1   = Zeilennummer High-Byte
                  Zelle Q2     = Vorwärtszeiger  Low-Byte
                  Zelle Q2+1   = Vorwärtszeiger High-Byte
```

Eine Textzeile lesen

```
22700 REM *** ENTSPRICHT SYS P4,T$ ***
22710 T$=""
22720 GET#FP,A$
22730 IFA$<>CHR$(13)THENT$=T$+A$:GOTO22720
22740 RETURN
```

```
Eingabeparameter: FP           = Logische Dateinummer der
                                 Eingabedatei
Ausgabeparameter: T$           = eingelesene Textzeile
```

Initialisierungsroutine

```
50500 REM *** ENTSPRICHT SYS PI ***
50510 HE$="0123456789ABCDEF"
50520 DEFFNV(X)=X-48+7*(X>64)
50530 TA=0
50540 POKEQ4,0
50550 DIMTN$(255),TV(255),TY(255)
50560 DIMBK$(255),K$(58)
50570 FORI=0TO127
50580 BK$(I)=CHR$(I)
50590 NEXT
50600 REM *** REST SPAETER BEI 50385
50610 RETURN
```

Bemerkung: Dieses Unterprogramm besetzt alle die Variablen und Felder, die nur in der reinen Basic-Version benötigt werden. Werden die oben beschriebenen Unterprogramme durch Maschinenprogramme ersetzt, so kann auch der Aufruf dieses Unterprogramms durch einen entsprechenden Aufruf einer Initialisierungsroutine für die Maschinenprogramme ausgetauscht werden.

3.2.2 Eine Zeile assemblieren

```
10000 REM * EINE ZEILE (T$) ASSEMBLIEREN                             *
10010 C$=""
10020 U$=" "
10030 GOSUB100
10040 A$=";":GOSUB700
10050 IFATHENT$=LEFT$(T$,A-1)
10060 GOSUB100
10070 IFLEFT$(T$,1)="%"THENGOSUB13000:RETURN
10080 A$="="
10090 GOSUB50
10100 IFA=0THEN10220
10110 T0$=LEFT$(T$,A-1)
10120 T$=MID$(T$,A+1)
10130 GOSUB100
10140 SI=2
10150 KM=0
10160 GOSUB15000
10170 T$=T0$
10180 GOSUB100
10190 T$=LEFT$(T$+SP$,8)
10200 TN$=T$:TV=W:TY=1:GOSUB18000
10210 RETURN
10220 IFAD$=""THENER=19:GOSUB17000:AD$=CHR$(0)+CHR$(0)
10230 A$=":":GOSUB700
10240 IFA=0THEN10300
10250 TN$=LEFT$(T$,A-1)
10260 T$=MID$(T$,A+1)
10270 GOSUB100
10280 TN$=LEFT$(TN$+SP$,8)
10290 TV=AD:TY=2:GOSUB18000
10300 A$=" "
10310 GOSUB50
10320 IFA<>4ANDLEN(T$)>3THENER=1:GOSUB17000:RETURN
10330 IFT$=""THENRETURN
10340 A$=LEFT$(T$,3):GOSUB600
10350 KM=KM(K)
10360 IFK=0THENER=1:GOSUB17000:RETURN
10370 T$=MID$(T$,4)
10380 GOSUB14000
10390 IFPEEK(Q4)THENRETURN
10400 IFMO>=0ANDKM=0THENER=7:GOSUB17000:RETURN
10410 IFMO=-1ANDKM>0THENER=3:GOSUB17000:RETURN
10420 ONKM+1GOTO11000,11100,11200,11300,11400,11500
10430 STOP
```

```
11000 M=0
11010 GOSUB12000
11020 RETURN
11100 M=2
11110 IFMO=1ORMO=4THENM=0
11120 IFMO=10THENM=1
11130 GOSUB12000
11140 HH=W:GOSUB250
11160 C$=C$+RIGHT$(HH$,2)+LEFT$(HH$,2)
11170 RETURN
11200 M=MO
11210 GOSUB12000
11220 IFMO=9THENRETURN
11230 H1=(MO=0ORMO=1ORMO=2ORMO=3ORMO=7ORMO=8)
11240 IFH1THENH=W:GOSUB200:C$=C$+H$:RETURN
11250 HH=W:GOSUB250
11270 C$=C$+RIGHT$(HH$,2)+LEFT$(HH$,2)
11280 RETURN
11300 IFW>255THENER=8:GOSUB17000:RETURN
11310 H=W:GOSUB200:C$=H$
11340 RETURN
11400 HH=W:GOSUB250
11420 C$=RIGHT$(HH$,2)+LEFT$(HH$,2)
11430 RETURN
11500 M=1
11510 IFMO=1ORMO=4THENM=0
11520 GOSUB12000
11530 H=W-AD-2
11540 IFH>127ORH<-128THENER=17:GOSUB17000:RETURN
11550 IFH<0THENH=H+256
11560 GOSUB200
11570 C$=C$+H$
11580 RETURN
```

Mit diesem Unterprogramm wird eine Zeile des Quelltextes in den entsprechenden Maschinencode umgewandelt. Weil in diesem Programmstück mehrere Funktionen vereinigt sind, wollen wir nacheinander auf die zugehörigen Programmteile eingehen.

Zunächst wird die Variable C$ annulliert, die später den Maschinencode in hexadezimaler Form aufnehmen soll. Dann wird der Merker U$, der das Auftreten von noch nicht definierten Symbolen anzeigen soll, mit einem Leerzeichen besetzt.

Bemerkung eliminieren

Im Quelltext beginnt eine Bemerkung immer mit einem Semikolon (;). Deshalb wird im Text ein solches Zeichen gesucht, und der Text ab der Stelle des Semikolons wird

abgeschnitten, da er für die Assemblierung nicht gebraucht wird.

Direktive feststellen

Ist das erste Zeichen des Quelltextes ein %-Zeichen, so liegt eine Direktive vor, und es wird das Unterprogramm zum Behandeln der Direktiven aufgerufen.

Konstanten-Definition

Eine Konstantendefinition hat die Form:

Symbol = Wert

Deshalb wird in diesem Programmstück zuerst ein Gleichheitszeichen im Quelltext gesucht, und, wenn keines gefunden wurde, zum Programmstück 'Marken' (siehe unten) übergegangen.

Zunächst werden die beiden Teilausdrücke vor und hinter dem Gleichheitszeichen in den Variablen T0$ und T$ zwischengespeichert und eventuell auftauchende Leerzeichen eliminiert. Dann werden die Variablen für die Größe des Symbolwertes und den Modus des Befehls gesetzt, die bei einer Konstanten-Definition immer SI=2 und KM=0 sind. Da die Konstantenvereinbarung aus einem Doppelterm (zwei Terme verknüpft mit einem einzigen Operator) bestehen kann, wird das Unterprogramm zur Auswertung dieses Ausdrucks aufgerufen. Schließlich wird das Symbol in die Symboltabelle eingefügt. Der Typ des Symbols ist hier '1', was bedeuten soll, daß dieses Symbol eine Konstante (keine Marke) ist.

Marken (engl.: Labels)

Eine Marke ist ein Symbol, welches den aktuellen Wert der Objektadresse übernimmt. Deshalb ist es notwendig, das spätestens an dieser Stelle die Startadresse des Objektprogramms definiert ist. Wenn nicht, wird eine Fehlermeldung ausgegeben.

Bei dem vorliegenden Assembler werden Markendefinitionen mit einem Doppelpunkt gekennzeichnet. Deshalb wird zunächst untersucht, ob die Zeile einen Doppelpunkt enthält, wenn nicht wird die Behandlung der Marken übersprungen.

Anschließend wird der Variablen TN$ der Name der Marke

zugeordnet, der Variablen T$ eine eventuell folgende Befehlszeile. Anschließend wird die Marke in die Symboltabelle eingefügt, wobei hier der Typ auf '2' gesetzt wird.

Typ und Modus einer Befehlszeile feststellen

Liegt weder eine Direktive noch eine Konstante noch eine Marke vor, so muß zwangsläufig eine Befehlszeile vorliegen. Das Programmstück von Zeile 10300 bis 10430 stellt den Modus und den Typ einer Befehlszeile fest und verzweigt in Zeile 10420 aufgrund des festgestellten Typs (Variable KM) in die entsprechenden Unterprogrammstücke.

Zunächst wird die gefundene mnemotechnische Bezeichnung in der Tabelle gesucht. Damit ergibt sich der Typ der Anweisung zu KM=KM(K), wenn K die Nummer der gefundenen Bezeichnung ist. Der Rest der Textzeile ist der Operand der Anweisung. Er wird an das Unterprogramm ab Zeile 14000 übergeben, welches den Adressierungsmodus (Variable MO) bestimmt. Die Werte von KM und MO haben folgende Bedeutung:

```
KM = 0  :  Der Befehl hat keinen Operanden
KM = 1  :  Der Befehl hat einen 2-Byte-Operanden
KM = 2  :  Der Befehl hat mehrere Adressierungs-
           möglichkeiten
KM = 3  :  Der Befehl hat einen 1-Byte-Operanden
           und hat keinen Befehlscode (BYT-Befehl)
KM = 4  :  Der Befehl hat einen 2-Byte-Operanden
           und hat keinen Befehlscode (WOR-Befehl)
KM = 5  :  Der Befehl ist ein relativer Sprung und hat
           demnach einen 1-Byte-Operanden

MO = -1 :  Kein Operand
MO =  0 :  Unmittelbare Adressierung ( #Op )
MO =  1 :  Zero-Page-Adressierung
MO =  2 :  Zero-Page-X-indiziert
MO =  3 :  Zero-Page-Y-indiziert
MO =  4 :  Absolute Adressierung
MO =  5 :  Absolut X-indizierte Adressierung
MO =  6 :  Absolut Y-indizierte Adressierung
MO =  7 :  Indiziert-indirekte Adressierung
MO =  8 :  Indirekt-indizierte Adressierung
MO =  9 :  Operand ist Akkumulator
MO = 10 :  Indirekte Adressierung
```

Befehle ohne Operand

Befehle ohne Operanden sind sehr einfach zu bearbeiten. Es braucht lediglich der Code des eigentlichen Befehls übergeben zu werden, was durch ein Unterprogrammaufruf ab Zeile 12000 erfolgt.

Absolute und indirekte Sprünge

Entsprechend der Sprungart wird die Variable M auf 0 (absolute Sprünge) oder auf 1 (indirekte Sprünge) gesetzt. Zeile 11130 setzt den entsprechenden Code in die Variable C$, und der Rest des Programmstückes bis hin zur Zeile 11160 wandelt den Operanden in eine hexadezimale Zahl um und ergänzt die Befehlszeile C$.

Befehle mit Operanden (keine Sprungbefehle)

Analog den eben dargestellten Programmstücken wird hier wieder entsprechend dem Modus der hexadezimale Wert des Befehls in die Variable C$ geschrieben und anschließend die in hexadezimale Zeichen umgerechneten Operanden.

Byte und Word

Die beiden Befehle BYT und WOR sind Konstanten-Definitionen für ein Byte bzw. zwei Byte. Diesen Konstanten wird zunächst keine Bezeichnung zugeordnet, sondern nur ein Wert. Z.B. werden so Variablen definiert, z.B. mit

'VAR1: BYT $00'

Relative Sprünge

In dem letzten Stück des Unterprogramms ab Zeile 11500 werden die relativen Sprünge behandelt, in dem auch hier wieder die Befehlszeile im Hexcode (C$) mit dem entsprechenden hexadezimalen Wert des Befehls und - als Operand - mit der Zahl der Bytes (im Hex-Code) besetzt wird, um die gesprungen werden soll. Diese Sprungart verringert die Rechenzeit. Wenn nur relative Sprünge verwendet werden, kann der Objektcode beliebig im Speicher ohne Auswirkung verschoben werden. Ein weiterer Grund ist die Speicherplatzersparnis, da relative Sprünge nur um maximal 127 Bytes nach vorne bzw. 128 Byte zurückspringen können, und damit der Operand in einem Byte untergebracht werden kann. Wurde für relativen Sprünge eine Sprungweite größer als

der eben angegebene Bereich errechnet, so wird eine Fehlermeldung ausgegeben.

ZEILE ASSEMBLIEREN 10000 - 11580

Variablen:

Name	Typ	Bereich	Bedeutung
A	R	0...255	Position von A$
A$	P	1 Zeichen	zu suchendes Zeichen
AA$	P	Zeichenreihe	zu suchende Zeichen
AD	G	0...65535	aktuelle Adresse
AD$	G	'' oder 2 Byte	Startadresse (L/H)
C$	A	bis 6 Zeichen	assemb. Maschinencode
ER	P	0...E9	Fehlernummer
H1	H	0 oder -1	logische Hilfsvariable
HH	P	0...65535	Wert für Hexzahl
K	R/P	0...K9	Nummer des Befehls
KM	P	0...5	Typ des Befehls
M	P	0...10	Adressierungsart
MO	R	-1...10	Adressierungsmodus
SI	P	1 oder 2	Anzahl Byte des Oper.
T$	E/P	Zeichenreihe	zu assemblier. Zeile
TO$	H	Zeichenreihe	Zeichenreihe bis '='
TN$	P	Zeichenreihe	Symbolname
TV	P/R	0...65535	Wert des Symbols
TY	P/R	0,1,2	Typ des Symbols
U$	A	' ' oder 'R'	'R' wenn Operand noch nicht definiert
W	R	0...65535	Wert des Doppelterms

Felder (Arrays):

Name	Dimen.	Typ	Bereich	Bedeutung
KM	58	G	0...5	Typ der Befehle

Unterprogrammaufrufe :

in	nach	Zweck
10030	100	Blanks von T$ eliminieren
10040	700	Position von A$ in T$
10060	100	Blanks von T$ eliminieren
10070	13000	Direktiven auswerten
10090	50	Position von A$ in T$
10130	150	Blanks von T$ eliminieren
10160	15000	Doppelterm auswerten
10180	100	Blanks von T$ eliminieren
10200	18000	Symbol in Tabelle einfügen
10220	17000	Fehler registieren
10230	700	Position von A$ in T$
10270	100	Blanks von T$ eliminieren
10290	18000	Symbol in Tabelle einfügen
10310	50	Position von A$ in T$
10320	17000	Fehler registieren
10340	600	Check auf Mnemonic Keyword
10360	17000	Fehler registieren
10380	14000	Modus feststellen
10400	17000	Fehler registieren
10410	17000	Fehler registieren
11010	12000	C$ mit Code besetzen
11130	12000	C$ mit Code besetzen
11150	250	4-stell. Hex-Zahl aus HH bilden
11210	12000	C$ mit Code besetzen
11240	200	2-stell. Hex-Zahl aus H bilden
11260	250	4-stell. Hex-Zahl aus HH bilden
11300	17000	Fehler registieren
11320	200	2-stell. Hex-Zahl aus H bilden
11410	250	4-stell. Hex-Zahl aus HH bilden
11520	12000	C$ mit Code besetzen
11540	17000	Fehler registieren
11560	200	2-stell. Hex-Zahl aus H bilden

Verzweigungen nach außen :

in Ze	nach	Bedingung	Bemerkung
10070	RETURN	LEFT$(T$,1)="%"	Direktive
10210	RETURN	'=' war in T$	Symbol eingefügt
10320	RETURN	A NE 4 AND	Kein Blank nach
		LEN(T$) GT 3	Befehl

```
!=============================================================!
! 10330 ! RETURN! T$=''               ! Zeile abgearbeitet !
! 10360 ! RETURN! K=0                 ! Ungültiger Befehl  !
! 10390 ! RETURN! PEEK(Q4) NE 0       ! Fehler beim Fest-  !
!       !       !                     ! stellen des Modus  !
! 10400 ! RETURN! MO GE 0  AND        ! Wenn KM=0,dann kein!
!       !       ! KM = 0              ! Operand erlaubt    !
! 10410 ! RETURN! MO = -1  AND        ! Operand fehlt,     !
!       !       ! KM GT 0             ! obwohl nötig       !
! 10430 ! STOP  ! KM GT 5             ! Wert von KM unmögl.!
! 11020 ! RETURN! KM = 0              ! kein Operand       !
! 11170 ! RETURN! KM = 1              ! JMP oder JSR-Befehl!
! 11220 ! RETURN! KM = 2 AND MO = 9   ! Operand = Akku     !
! 11240 ! RETURN! H1                  ! Ein-Byte-Operand   !
! 11280 ! RETURN! KM = 2 AND NOT H1   ! Zwei-Byte-Operand  !
! 11300 ! RETURN! KM = 3   AND        ! Byte größer als 255!
!       !       ! W GT 255            !                    !
! 11340 ! RETURN! KM = 3              ! Byte reserviert    !
! 11430 ! RETURN! KM = 4              ! Word reserviert    !
! 11540 ! RETURN! H GT 127  OR        ! Relativer Sprung   !
!       !       ! H LT -128           ! zu weit            !
! 11580 ! RETURN! KM = 5              ! Relativer Sprung   !
!=============================================================!
```

3.2.3 Variable mit hexadezimalem Code besetzen

```
12000 REM * C$ MIT CODE BESETZEN                              *
12010 C$=KC$(K,M)
12020 IFC$=""THENER=6:GOSUB17000
12030 RETURN
```

In diesem kurzen Unterprogramm wird in der Variablen C$ der hexadezimale Befehlscode eingetragen. Dazu wird aus den in Kapitel 3.1 beschriebenen DATA-Statements lediglich der entsprechende Code aus der Variabeln KC$ (K,M) übernommen. Wurde kein Code gefunden, so wird eine Fehlermeldung ausgedruckt.

3.2.4 Direktiven auswerten

```
13000 REM * %DIRECTIVEN AUSWERTEN                              *
13020 GOSUB24000
13030 ONDGOTO13100,13200,13300,13400,13500,13600
13040 ER=2:GOSUB17000
13060 RETURN
```

Zunächst wird in der Variablen AA$ der Name der Direktive (der auch mehr als drei Zeichen enthalten kann) abgelegt und mit dem Unterprogramm ab Zeile 24000 die Nummer der Direktive bestimmt. Aufgrund dieser Nummer wird anschliessend verzweigt. Wenn nicht verzweigt wird, so wird eine Fehlermeldung ausgegeben.

%ORG

```
13100 REM *** %ORG DIRECTIVE ***
13105 IFAD$>""THENER=20:GOSUB17000:RETURN
13110 T$=MID$(T$,5)
13115 KM=4
13120 GOSUB100
13125 GOSUB16000
13130 IFPEEK(04)THENRETURN
13135 AD=W
13140 HH=INT(AD/256)
13145 H=AD-256*HH
13150 AD$=CHR$(H)+CHR$(HH)
13155 PRINT#3,AD$;
13160 SA=AD
13165 RETURN
```

Die ersten zwei Byte der Objekt-Datei werden in der Variablen AD$ abgespeichert. Wenn schon eine Startadresse definiert wurde, so wird ein Fehler ausgegeben. Sonst wird in den Zeilen 13110 bis 13135 der Operand ausgewertet und in der Variablen AD abgelegt. In dem restlichen Programmstück wird die Dezimalzahl noch in zwei Zeichen zerlegt, die jeweils das Lower und Higher Byte angeben. An dieser Stelle sei noch angemerkt, daß das Speichern in der Objekt-Datei (auf Floppy) jeweils in zwei zusammengefaßten Hexadezimalziffern zu einem Zeichen gespeichert wird. Z.B. wird hexadezimal 'FF' als CHR$ (255) gespeichert.

Als Abschluß wird noch in der globalen Variablen SA der Wert der Startadresse festgehalten, um ihn im weiteren Programm verwenden zu können.

%END

```
13200 REM *** %END DIRECTIVE ***
13210 IS=320
13220 RETURN
```

In dieser Zeile wird lediglich der Eingabestatus IS als Merker für das Ende der Quelldatei gesetzt.

%INO

```
13300 REM *** %INO DIECTIVE ***
13305 IFAD$<>""THENER=20:GOSUB17000:RETURN
13310 KM=4
13315 INPUT"▒STARTADRESSE     ████";T$
13320 IFT$="  "THEN13315
13325 GOSUB16000
13330 IFPEEK(Q4)THENRETURN
13335 AD=W
13340 HH=INT(AD/256)
13345 H=AD-256*HH
13350 AD$=CHR$(H)+CHR$(HH)
13355 PRINT#3,AD$;
13360 SA=AD
13365 RETURN
```

Dieses Programmstück funktioniert analog der oben beschriebenen %ORG-Direktive, außer daß die Startadresse nicht als Operand in der Quelldatei steht, sondern vom Bildschirm eingelesen wird.

%DUP

```
13400 REM *** %DUP DIRECTIVE ***
13404 T$=MID$(T$,5)
13408 A$=",":GOSUB700
13412 IFA=0THENER=3:GOSUB17000:RETURN
13416 T3$=MID$(T$,A+1)
13420 T$=LEFT$(T$,A-1)
13424 SI=2
13428 KM=0
13432 GOSUB15000
13436 AN=W
13440 T$=T3$
13444 GOSUB10000
13448 IFAN=0THENC$=""RETURN
13452 IFAN=1THENRETURN
13456 CC$="":A=0
13460 IFA>=LEN(C$)THEN13476
13464 HH$=MID$(C$,A+1,2):GOSUB400:CC$=CC$+CHR$(HH)
13468 A=A+2
```

```
13472 GOTO13460
13476 FORI=1TOAN-1
13480 PRINT#3,CC$;
13484 AD=AD+LEN(CC$)
13488 NEXT
13492 RETURN
```

Die %DUP-Direktive hat die Form:

%DUP Anzahl, Anweisung

Deshalb wird im Text hinter dem Wort '%DUP' ein Komma gesucht. Wenn keines vorhanden ist, wird ein Fehler gemeldet. Der Text wird dann zerlegt in einen Teil vor dem Komma (T$) und den Teil dahinter (T3$). T$ wird an das Unterprogramm zum Auswerten eines Doppelausdrucks übergeben und das Ergebnis in der Variablen AN gespeichert, die dann die Anzahl der Wiederholungen enthält. T3$ wird nun in T$ übertragen und dem Unterprogramm zum Assemblieren einer Zeile übergeben, womit in C$ der fertig assemblierte Code der Anweisung steht. In C$ ist der Code jedoch in hexadezimaler Form enthalten, so daß er noch in die Variable CC$ als Zeichenreihe kopiert wird. Dieser Code wird (AN-1)-mal in die Objektdatei geschrieben, wobei jeweils der Adresszähler um die Länge des Codes CC$ erhöht wird. Das Code wird auf jeden Fall an das aufrufende Unterprogramm zurückgegeben, wo nachher die letzte Speicherung erfolgt.

Das Unterprogramm ist sowohl aufrufendes Programm, es wird aber auch von hier aufgerufen. Dies nennt man einen indirekt rekursiven Aufruf. Man muß bei einer derartigen Programmierweise darauf achtgeben, daß die Parameter nicht durch einen neuen Aufruf des Unterprogramms überschrieben werden und daß der Unterprogramm-Stapel nicht überläuft. Im vorliegenden Fall muß der Anwender dafür sorgen, daß die Anweisung, die wiederholt werden soll nicht selbst eine Direktive ist. Der Fall einer weiter verschachtelten Rekursion wird dadurch automatisch vermieden.

%ASC

```
13500 REM *** %ASC DIRECTIVE ***
13505 T$=MID$(T$,5)
13510 A$=CHR$(34):GOSUB700
13515 IFA=0THENER=3:GOSUB17000:RETURN
13520 T$=MID$(T$,A+1)
13525 A$=CHR$(34):GOSUB700
13530 IFATHENT$=LEFT$(T$,A-1)
13535 C$=""
13540 IFT$=""THENRETURN
13545 H=ASC(RIGHT$(T$,LEN(T$))):GOSUB200
13550 C$=H$+C$
13555 T$=LEFT$(T$,LEN(T$)-1)
13560 IFLEN(C$)<6THEN13540
13565 PRINT#3,T$;
13570 AD=AD+LEN(T$)
13575 RETURN
```

Die %ASC-Direktive wird benützt, um Text in die Objektdatei einzufügen. Die Syntax des Befehls ist:

%ASC "Text"

Im Operanden werden zwei "-Zeichen gesucht, und der dazwischenliegende Text steht anschließend in T$. Die letzten drei Zeichen werden in Hexadezimalziffern umgewandelt und als Variable C$ dem aufrufenden Programm zurückgegeben, der Rest des Textes wird direkt in die Objektdatei geschrieben, wobei natürlich der Adresszähler entsprechend erhöht wird.

%INCLUDE

```
13600 REM * %INCLUDE-DIRECTIVE                        *
13605 A$=CHR$(34):GOSUB700
13610 IFA=0THENER=24:GOSUB17000:RETURN
13615 T$=MID$(T$,A+1,17)
13620 A$=CHR$(34):GOSUB700
13625 IFA=0THENER=24:GOSUB17000:RETURN
13630 T$=LEFT$(T$,A-1)
13635 OPEN6,8,6,T$
13640 GOSUB25000
13645 IFDSTHENER=23:GOSUB17000:CLOSE6:RETURN
13650 FP=6:AS=IS
13655 IFFT=2THENGOSUB10:GOSUB10
13660 RETURN
```

Mit der %INCLUDE-Direktive kann eine andere Quelldatei in den Quelltext eingebunden werden. Dazu wird der Name der einzufügenden Datei bestimmt, diese Datei mit der logischen Nummer 6 geöffnet und die Variable FP ebenfalls auf 6 gesetzt, womit erreicht wird, daß nun aus dieser Datei anstatt aus der normalen Quelldatei gelesen wird. Der Einlesestatus der normalen Quelldatei muß in AS gesichert werden, weil die %INCLUDE-Direktive ja die letzte Anweisung des Quellprogramms sein könnte.

Im Unterprogramm ab Zeile 22000 wird dafür gesorgt, daß die Eingabevariable FP wieder auf den Normalwert 2 gesetzt wird, wenn die einzufügende Datei zu Ende ist.

DIREKTIVEN AUSWERTEN 13000 - 13660

Variablen:

Name	Typ	Bereich	Bedeutung
A	R/H	0...255/0...65535	Position / Zähler
AD	G	0...65535	aktuelle Adresse
AD$	G	2 Zeichen (Byte)	Startadresse kodiert
AN	H	0...65535	Anzahl Wiederholungen
AS	G	0...255	Alter Eingabestatus
C$	G	Zeichenreihe	Assemblierter Code
CC$	H	Zeichenreihe	Assemblierter Code
DS	R	0..74	Disk-Status
ER	P	0..E9	Fehlernummer
FP	G	2,6	Log. Nr. Eingabedatei
FT	G	1,2	Typ d. Eing.dat. (s,p)
H	H/R/P	0...255	Lower-Byte Startadr.
HH	H/R	0...255	Higher-Byte Startadr.
H$	P/R	2 Zeichen	Lower-Byte Startadr.
HH$	P	4 Zeichen	Higher-Byte Startadr.
I	H	0...AN-1	Laufvariable
IS	A	0...320	Input-Status
KM	P	0...5	Typ des Befehls
SA	G	0...65535	Startadresse
SI	P	0,1,2	Größe des Operanden
T$	P	Zeichenreihe	Operand
T3$	H	Zeichenreihe	zu wiederh. Befehl
W	R	0...65535	Wert von Doppelausdr.

Dateien :

#	Name	T	Bemerkung
3	F$+"...."	p	Vorläufige Objektdatei
6	T$	p/s	Einzubindende Quelldatei

Unterprogrammaufrufe :

in	nach	Zweck
13020	24000	Nummer der Direktive feststellen
13040	17000	Fehler registrieren
13105	17000	Fehler registrieren
13120	100	Blanks eliminieren
13125	16000	Einzelausdruck auswerten
13305	17000	Fehler registrieren
13325	16000	Einzelausdruck auswerten
13408	700	Zeichen suchen
13412	17000	Fehler registrieren
13432	16000	Doppelausdruck auswerten
13444	10000	Zeile assemblieren
13464	400	Wert von Hexzahl bestimmen
13510	700	Zeichen suchen
13515	17000	Fehler registrieren
13525	700	Zeichen suchen
13545	200	2-stellige Hexzahl bilden
13605	700	Zeichen suchen
13610	17000	Fehler registrieren
13620	700	Zeichen suchen
13625	17000	Fehler registrieren
13640	25000	Disk-Status-Werte bestimmen
13645	17000	Fehler registrieren
13655	10	Ein Zeichen aus Datei #FP lesen

Verzweigungen nach außen :

in Ze	nach	Bedingung	Bemerkung
13060	RETURN	D=0	keine Direktive
13105	RETURN	AD$ GT ""	Startadresse bereits definiert
13130	RETURN	PEEK(Q4) NE 0	Fehler im Einzel-

```
!===========================================================!
!        !         !                      ! term            !
! 13165  ! RETURN! D=1                    ! Ende von %ORG   !
! 13220  ! RETURN! D=2                    ! Ende von %END   !
! 13305  ! RETURN! AD$ GT ""              ! Startadresse    !
!        !         !                      ! bereits definiert !
! 13330  ! RETURN! PEEK(Q4) NE 0          ! Fehler im Einzel- !
!        !         !                      ! term            !
! 13365  ! RETURN! D=3                    ! Ende von %INO   !
! 13412  ! RETURN! A=0                    ! Kein "," bei %DUP !
! 13412  ! RETURN! AN=1                   ! Anzahl=1 bei %DUP !
! 13492  ! RETURN! D=4                    ! Ende von %DUP   !
! 13515  ! RETURN! A=0                    ! Kein " bei %ASC !
! 13540  ! RETURN! T$=""                  ! Text leer       !
! 13575  ! RETURN! D=5                    ! Ende von %ASC   !
! 13610  ! RETURN! A=0                    ! Kein ","        !
! 13625  ! RETURN! A=0                    ! Kein ","        !
! 13645  ! RETURN! DS NE 0                ! Diskfehler bei OPEN!
! 13660  ! RETURN! D=6                    ! Ende von %INCLUDE !
!===========================================================!
```

3.2.5 Modus und Wert des Operanden feststellen

```
14000 REM * MODE FESTSTELLEN                                    *
14010 REM MODE MO : -1=NO OPER.;  0=# USW. S.ZEILE 54030
14020 REM MODE MO = 10 : INDIRECT
14030 REM SIZE:  SI=1 -> BYTE      SI=2 -> WORD
14040 GOSUB100
14050 IFT$=""THENMO=-1:RETURN
14060 IFT$="A"THENMO=9:RETURN
14070 AA$=CHR$(34)+"#(),.@!":GOSUB800
14080 IFA(2)<>-(A(3)>0)THENER=4:GOSUB17000:RETURN
14090 L=LEN(T$)
14100 XY=(RIGHT$(T$,1)<>"X"ANDRIGHT$(T$,1)<>"Y")
14110 IFA(4)THENIFA(5)>0ORA(4)<>L-1ORXYTHEN14410
14120 IFA(5)THENIFA(4)>0ORA(5)<>L-1ORXYTHEN14410
14130 IFA(1)>1ORA(6)>1ORA(7)>1THENER=5:GOSUB17000:RETURN
14140 IFA(1)THENIFA(2)+A(3)+A(4)+A(5)+A(6)+A(7)THEN14410
14150 IFA(6)THENIFA(7)THEN14410
14160 IFA(7)THENIFA(6)THEN14410
14170 IFA(6)+A(7)THENIFA(1)+A(2)+A(3)THEN14410
14180 IFA(1)THENMO=0:T$=MID$(T$,2):GOTO14440
14190 IFRIGHT$(T$,3)=",X)"THENMO=7:GOTO14430
14200 IFRIGHT$(T$,3)=".X)"THENMO=7:GOTO14430
14210 IFRIGHT$(T$,3)="),Y"THENMO=8:GOTO14430
14220 IFRIGHT$(T$,3)=").Y"THENMO=8:GOTO14430
14230 IFRIGHT$(T$,1)=")"THENMO=10:T$=MID$(T$,2,L-2):SI=2
                                        :GOSUB15000:RETURN
```

```
14240 ZP=0
14250 SI=2
14260 IFA(6)THENZP=1:T$=MID$(T$,2):L=L-1:SI=1
14270 IFA(7)THENZP=2:T$=MID$(T$,2):L=L-1
14280 IFRIGHT$(T$,2)=",X"THENMO=5:GOSUB14370:GOTO14340
14290 IFRIGHT$(T$,2)=".X"THENMO=5:GOSUB14370:GOTO14340
14300 IFRIGHT$(T$,2)=",Y"THENMO=6:GOSUB14370:GOTO14340
14310 IFRIGHT$(T$,2)=".Y"THENMO=6:GOSUB14370:GOTO14340
14320 MO=4
14330 GOSUB14380
14340 IFPEEK(Q4)THENRETURN
14350 IFW<256ANDZP<2THENMO=MO-3
14360 RETURN
14370 T$=LEFT$(T$,L-2)
14380 IFZP=1THENMO=MO-3
14390 GOSUB15000
14400 RETURN
14410 ER=10:GOSUB17000
14420 RETURN
14430 T$=MID$(T$,2,L-4)
14440 SI=1
14450 GOSUB15000
14460 RETURN
```

Dieses Unterprogramm dient letztendlich zum Besetzen der Variablen MO,SI und W, in welchen der Adressierungsmodus der Anweisung, sowie die Größe und der Wert des Operanden festgehalten werden. Nicht festgestellt wird in diesem Unterprogramm, ob der jeweilige Befehl auch diesen Modus kennt.

Die Bedeutungen der verschiedenen Werte von MO sind im Kapitel 3.2.2 tabellarisch aufgeführt. Der Modus wird aufgrund von auftretenden Sonderzeichen und der Größe der Operanden festgestellt. Die Größe des Operanden (SI) kann folgende Werte annehmen:

SI=1 Operanden bis 255 (1-Byte-Operanden)
SI=2 Operanden bis 65535 (2-Byte-Operanden)

Ist der Operand eine leere Zeichenreihe, so wird MO mit -1 besetzt , ist der Operand ein 'A' wird MO mit 9 besetzt. In beiden Fällen wird das Unterprogramm sofort verlassen.

Die Sonderzeichen werden mit Hilfe des Unterprogramms ab Zeile 800 lokalisiert. Die Zeilen 14080 bis 14160 enthalten diverse Prüfungen auf die Richtigkeit der Syntax (sind Klammern richtig gesetzt,...).

Eine wichtige Bedeutung hat die Variable ZP. Hier wird festgehalten, ob durch ein Sonderzeichen im Operanden die Zero-Page oder die absolute Adressierung erzwungen wird.

Die Variable wird nur besetzt, wenn der Modus durch bestimmte Sonderzeichen-Kombinationen (z.B. ",X)") noch nicht ermittelt wurde. Es bedeuten:

ZP=0 Zwischen Zero-Page- und absoluter Adressierung kann allein aufgrund der Größe des Operanden entschieden werden.

ZP=1 Das erste Zeichen des Operanden ist ein 'Klammeraffe'. Dann kann nur Zero-Page Adressierung ausgewählt werden. Ist der Operand aber doch größer als 255, so wird später eine Fehlermeldung ausgegeben.

ZP=2 zeigt ein Rufzeichen an der ersten Stelle im Operanden an. Dadurch wird die absolute Adressierung erzwungen. Das ist manchmal notwendig, wenn man in der Zero-Page Y-indiziert arbeiten möchte, der Befehl diese Adressierungsart jedoch nicht kennt.

Mit diesen Informationen kann in jedem Fall die zugehörige Adressierungsart bestimmt werden. Details hierzu entnehmen Sie bitte dem Listing.

Mode feststellen 14000 - 14460

Variablen:

Name	Typ	Bereich	Bedeutung
A	R	0...255	Position von A$ in AA$
A$	P	1 Zeichen	zu suchendes Zeichen
AA$	P	Zeichenreihe	zu suchende Zeichen
ER	P	0...E9	Fehlernummer
L	H	0...79	Länge von T$
MO	A	-1...10	Modus:
			-1 = kein Operand
			0 = unmittelbar
			1 = Zero-Page
			2 = Zero-Page,x
			3 = Zero-Page,y
			4 = absolut
			5 = absolut,x
			6 = absolut,y
			7 = vor -indiziert
			8 = nach-indiziert
			9 = Akkumulator

			10 = indirect
Q4	G	49208	Adr. Fehleranzahlzelle
SI	A	1 oder 2	Anzahl Bytes des Operanden
T$	E/P/A	Zeichenreihe	Operand
W	R	0...65535	Wert des Doppelterms
XY	H	0 oder -1	XY=0,wenn letztes Zei. d. Oper. 'x' od. 'y'
ZP	H	0...2	Zeropage-Option 0=keine Vorgabe 1=Vorgabe : Zeropage 2=Vorgabe : Absolute

Felder (Arrays):

Name	Dimen.	Typ	Bereich	Bedeutung
A	20	R	0...255	Positionen der Sonderz. aus AA

Unterprogrammaufrufe :

in	nach	Zweck
14040	100	Blanks von T$ eliminieren
14070	800	Positionen von Zeichen in T$ bestimmen
14080	17000	Fehler registrieren
14130	17000	Fehler registrieren
14230	15000	Doppelausdruck auswerten
14390	15000	Doppelausdruck auswerten
14410	17000	Fehler registrieren
14450	15000	Doppelausdruck auswerten

Verzweigungen nach außen :

in Ze	nach	Bedingung	Bemerkung
14050	RETURN	T$=""	Kein Operand
14050	RETURN	T$="A"	Operand = Akku
14080	RETURN	a(2) NE -(a(3) GT 0)	Klammern nicht paarweise
14130	17000	a(1) GT 1 OR a(6) GT 1 OR	Immediate- bzw. Zeropagezeichen

```
!==========================================================!
!       !        ! a(7) GT 1          ! nicht an 1.Stelle  !
! 14230 ! RETURN! RIGHT$(T$,1)=")"   ! Indirekte Adress.  !
! 14340 ! RETURN! PEEK(Q4) NE 0      ! Fehler             !
! 14360 ! RETURN!                    ! Ende               !
! 14400 ! RETURN! div. Möglichkeiten! Fehlernummer 10    !
! 14460 ! RETURN!                    ! Ende               !
!==========================================================!
```

3.2.6 Doppelterm auswerten

```
15000 REM * DOPPELAUSDRUCK AUSWERTEN                          *
15010 AA$=CHR$(34)+"#(),.@!+-*/$↑":GOSUB800
15020 IFA(1)+A(2)+A(3)+A(4)+A(5)+A(6)+A(7)THEN15220
15030 A=-(A(8)>0)-(A(9)>0)-(A(10)>0)-(A(11)>0)
15040 IFA=0THENGOSUB16000:GOTO15200
15050 IFA>1THENER=12:GOSUB17000:RETURN
15060 A=A(8)+A(9)+A(10)+A(11)
15070 T1$=LEFT$(T$,A-1)
15080 T2$=MID$(T$,A+1)
15090 T$=T1$
15100 GOSUB16000
15110 W1=W
15120 T$=T2$
15130 GOSUB16000
15140 W2=W
15150 IFPEEK(Q4)THENRETURN
15160 IFA(8)THENW=W1+W2
15170 IFA(9)THENW=W1-W2
15180 IFA(10)THENW=W1*W2
15190 IFA(11)THENW=INT(W1/W2)
15200 IFW>=256↑SITHENER=13:GOSUB17000:RETURN
15210 RETURN
15220 ER=10:GOSUB17000
15230 RETURN
```

Unter Doppelterm verstehen wir beim Assembler Terme, die aus zwei Operanden und einer Verknüpfung (+,-,*,/) bestehen. In diesem Unterprogramm selbst werden Doppelterme ausgewertet. Jedoch werden können auch einzelne Terme ausgewertet werden, da am Beginn eine entsprechende Verzweigung zu dem Unterprogramm ab Zeile 16000 vorhanden ist.

In Zeile 15010 werden die Positionen der Sonderzeichen bestimmt. Folgende Zeichen dürfen nicht auftreten: ", #, (,), Komma, ., Klammeraffe, !. Wenn eines dieser Zeichen im Doppelausdruck enthalten ist, wird eine Fehlermeldung

ausgegeben, ebenso, wenn mehr als ein Verknüpfungszeichen auftritt.

Wenn kein Sonderzeichen erscheint, liegt ein Einzelausdruck vor, das entsprechende Unterprogramm wird aufgerufen, und es wird zum Ende des Unterprogramms verzweigt.

Zuerst wird der Doppelausdruck zerlegt. Die Variablen T1$ (linker Teil) und T2$ (rechter Teil) beinhalten anschließend die durch das Sonderzeichen getrennten Textteile. Beide Ausdrücke werden durch Aufrufen des Unterprogramms für Einzelausdruck auswerten getrennt behandelt.

In den Zeilen 15160 bis 15190 werden die eigentlichen Verknüpfungen durchgeführt, wobei die in den Variablen W1 und W2 festgehaltenen Dezimalwerte der Operanden verwendet werden. Die Division wird als ganzzahlige Division ausgeführt.

Doppelausdruck auswerten 15000 - 15230

Variablen:

Name	Typ	Bereich	Bedeutung
A	H	0...255	Position des Verküpfungszeichens
AA$	P	Zeichenreihe	zu suchende Zeichen
ER	P/R	0...E9	Fehlernummer
SI	E	1 oder 2	Anzahl Bytes des Operanden
T$	E/P/A	Zeichenreihe	Operand
T1$	H	Zeichenreihe	Erster Term
T2$	H	Zeichenreihe	Zweiter Term
W	R/A	0...65535	Wert des Einzelterms Wert des Doppelterms
W1	H	0...65535	Wert des 1.Terms
W2	H	0...65535	Wert des 2.Terms

Felder (Arrays):

Name	Dimen.	Typ	Bereich	Bedeutung
A	20	R	0...255	Position jedes Sonderz. von AA$

Unterprogrammaufrufe :

in	nach	Zweck
15010	800	Positionen von Zeichen in T$ bestimmen
15040	16000	Einzelausdruck auswerten
15050	17000	Fehler registrieren
15100	16000	Einzelausdruck auswerten
15130	16000	Einzelausdruck auswerten
15200	17000	Fehler registrieren
15220	17000	Fehler registrieren

Verzweigungen nach außen :

in Ze	nach	Bedingung	Bemerkung
15050	RETURN	A GT 1	Mehr als 1 Operator
15150	RETURN	PEEK(Q4) NE 0	Fehler
15210	RETURN		Ende
15230	RETURN	A(1...7) NE 0	Falsches Sonderz. im Doppelterm

3.2.7 Einzelausdruck auswerten

```
16000 REM * EINZELAUSDRUCK AUSWERTEN                          *
16010 GOSUB100
16020 HW=0
16030 IFLEFT$(T$,1)="<"THENHW=1:T$=MID$(T$,2)
16040 IFLEFT$(T$,1)=">"THENHW=2:T$=MID$(T$,2)
16050 IFT:$="$"THENW=AD:GOTO16510
16060 IFLEFT$(T$,1)="%"THENT$=MID$(T$,2):GOTO16110
```

```
16070 IFLEFT$(T$,1)<>"$"THEN16150
16080 T$=MID$(T$,2)
16090 IFLEN(T$)<5THENHH$=T$:GOSUB400:W=HH:GOTO16510
16100 IFLEN(T$)<>8THENER=14:GOSUB17000:RETURN
16110 BB$=T$
16120 GOSUB23000
16130 W=BB
16140 GOTO16510
16150 IFASC(T$)>=48ANDASC(T$)<=57THENW=VAL(T$):GOTO16510
16160 T$=LEFT$(T$+SP$,8)
16170 TN$=T$:GOSUB500:W=TV
16180 IFTYTHEN16510
16190 UA=AD+1
16200 UN$=T$
16210 ONKMGOTO16250,16300,16370,16400,16430
16220 ER=16:GOSUB17000
16230 UT=0
16240 GOTO16460
16250 IFMO=1ORMO=4THENUT=7:GOTO16450
16260 IFMO=10THENUT=6:GOTO16450
16270 ER=6:GOSUB17000
16280 UT=0
16290 GOTO16460
16300 IFMO=0THENUT=1:GOTO16450
16310 IFMO<4THENUT=3:GOTO16450
16320 IFMO<=6THENUT=5:GOTO16450
16330 IFMO<=8THENUT=4:GOTO16450
16340 ER=6:GOSUB17000
16350 UT=0
16360 GOTO16460
16370 UT=1
16380 UA=AD
16390 GOTO16450
16400 UT=2
16410 UA=AD
16420 GOTO16450
16430 UT=8
16440 GQTO16450
16450 GOSUB19000
16460 U$="R"
16470 W=32768+128:REM DEFAULT WERT FUER ADRESSEN $8080
16480 IFUT=1ORUT=3ORUT=4THENW=128:REM DEFWERT FUER BYTEW.
16490 IFUT=8THENW=AD+2:REM DEFWERT FUER BRANCHES
16500 REM
16510 IFHW=1THENW=W-256*INT(W/256)
16520 IFHW=2THENW=INT(W/256)
16530 RETURN
```

Zunächst werden in dem Unterprogramm zum Auswerten eines Einzelausdrucks auch wieder die führenden und folgenden Leerzeichen abgeschnitten. Wie bereits im letzten Kapitel erwähnt, wird dieses Unterprogramm bei Doppeltermen auch ausgewertet, dann jedoch für jeden Teil des Terms getrennt.

Wenn der Term mit einem Kleiner- oder einem Größerzeichen beginnt, so darf nur das Low-Byte oder das High-Byte des Wertes als Ergebnis zurückgegeben werden. Zu Beginn des Unterprogramms wird deshalb der Merker HW (=Halbwort) gesetzt. Folgende Werte sind möglich:

HW=0   Wert ist Wert des gesamten Terms
HW=1   Wert ist Low-Byte des Termwertes
HW=2   Wert ist High-Byte des Termwertes

Wenn der Einzelterm nur aus einem '$' besteht, so wird die Adresse der aktuellen Programmzeile als Wert herangezogen. Liegen hinter dem '$' noch mehr Zeichen vor, so steht dort entweder ein hexadezimaler oder ein binärer Ausdruck. Ob hexadezimal oder binär, ergibt sich aus der Länge des dem $-Zeichen folgenden Ausdruckes: acht weitere Zeichen bedeuten immer binär; bis vier Zeichen immer hexadezimal. Ist das erste Zeichen ein "%" so wird ebenfalls binär ausgewertet.

Wenn der Operand eine Ziffer ist, so wird dessen Wert einfach über die Funktion VAL bestimmt. In allen anderen Fällen, liegt mit Sicherheit ein Symbol vor, das dann im weiteren in der Symboltabelle herausgesucht wird.

Wenn das Symbol in der Tabelle gefunden wurde, so wird der dort gespeicherte Wert der Variablen W zugeordnet, und das Unterprogramm verlassen.

In den Zeilen 16190 bis 16490 erfolgt die Behandlung für undefinierte Symbole. Es tritt öfters der Fall ein, daß ein Symbol an der Stelle, wo es angesprochen wird, noch nicht definiert ist, z.B. bei Vorwärtssprüngen. Der Assembler kann natürlich den wahren Wert des Symbols nicht in die Objektdatei eintragen, andererseits muß er den übrigen Quelltext weiterverarbeiten können. Dieses Problem der Vorwärtsverweise wird im vorliegenden Assembler folgendermaßen gelöst:

Jedes noch nicht definierte Symbol wird in eine Tabelle der undefinierten Symbole eingefügt. Anstatt des wahren Wertes wird ein Vorgabewert (engl.: default value) in die Objektdatei eingetragen. Im Protokoll wird ein "R" eingetragen, womit dem Leser des Protokolls angezeigt wird, daß

die in der Spalte 'Code' angegebenen Werte nur Vorgabewerte sind.

Nachdem das gesamte Quellprogramm bearbeitet ist, müßten alle Symbole definiert sein. Der Assembler liest dann die eben erstellte vorläufige Objektdatei und ersetzt an den betreffenden Stellen den Vorgabewert durch den wahren Wert. Die Tabelle der undefinierten Symbole beinhaltet deswegen ein Feld UA(), in dem die Adressen gespeichert sind, bei welchen die wirklichen Werte eingesetzt werden müssen und ein Feld UN$(), in dem die Namen der undefinierten Symbole abgelegt sind.

In der Tabelle der undefinierten Symbole wird auch ein Feld UT() mitgeführt, das die Art des Operanden angibt, bei dem das undefinierte Symbol gebraucht wurde. Daraus läßt sich auch die Zahl der zu ändernden Bytes (1 oder 2) bestimmen. Folgende Werte für eine Variable UT() bzw. UT sind möglich:

UT=0	Typ des undefinierten Symbols noch nicht festgestellt
UT=1	Bytewert für BYT-Befehl oder #-Adressierung
UT=2	Wert für WOR-Befehl
UT=3	Zero-Page-Adresse
UT=4	Zero-Page-Adresse eines Vektors
UT=5	Absolute Adresse
UT=6	Adresse des Vektors für indirekten Sprung
UT=7	absolutes Sprungziel
UT=8	relatives Sprungziel

Nach diesem Exkurs über die Tabelle der undefinierten Symbole betrachten wir wieder das Unterprogramm. Die Zeile 16190 wird nur erreicht, wenn ein undefiniertes Symbol auftrat. Hier wird in der Variablen UA die Adresse des undefinierten Symbols festgehalten und in der Variablen UN$ das Symbol. Im weiteren muß nur noch der Typ des undefinierten Symbols bestimmt werden. Abhängig vom Typ (KM) des Befehls wird in Zeile 16210 zu unterschiedlichen Programmstücken verzweigt. Ist KM=0, so wird das Symbol in einer Konstantenvereinbarung oder bei einer Direktive benötigt, wo es unbedingt definiert sein muß. Deshalb wird in diesem Fall eine Fehlermeldung ausgegeben.

In den Zeilen 16250 bis 16440 wird abhängig von Typ und Modus der Anweisung die Variable UT besetzt. In den Zeilen 16450 bis 16500 wird schließlich das undefinierte Symbol in die Tabelle eingetragen, die Variable U$ auf 'R' gesetzt, und die Variable W mit einem Default-Wert, abhängig vom Typ des undefinierten Symbols, besetzt.

Ab Zeile 16500 steht das Programmstück, welches den Wert des Einzelausdrucks entsprechend dem Merker HW (s.o.) zurechtschneidet.

Einzelausdruck auswerten 16000 - 16530

Variablen:

Name	Typ	Bereich	Bedeutung
AD	G	0...65535	aktuelle Adresse
BB	R	0...255	Wert von Binär-Zahl
BB$	P	Zeichenreihe	Binär-Zahl
ER	P	0...E9	Fehlernummer
HH	R	0...65535	Wert v. 4-stell. Hex-Z
HH$	P	4 Zeichen	4-stellige Hex-Zahl
HW	H	0,1,2	Merker f. Halbwortzei.
KM	E	0...5	Typ des Befehls
MO	E	-1...10	Adressierungsmodus
T$	E	Zeichenreihe	Einzelterm
TY	R	0,1,2	Typ des Symbols
TN$	P	Zeichenreihe	Gesuchtes Symbol
TV	R	0...65535	Wert des Symbols
U$	A	' ' oder 'R'	'R' bei undef. Symbol
UA	P	0...65535	Adresse, bei der un-def. Symbol auftrat
UN$	P	Zeichenreihe	Name d. undef. Symbols
UT	P	0...8	Typ des undef. Symbols
W	A	0...65535	Wert des Einzelterms

Unterprogrammaufrufe :

in	nach	Zweck
16010	100	Blanks eliminieren
16090	400	Wert von Hexadezimalzahl bestimmen
16100	17000	Fehler registrieren
16120	23000	Wert von 8-stell. Binärzahl bestimmen
16170	500	Symbol in Tabelle suchen
16220	17000	Fehler registrieren
16270	17000	Fehler registrieren
16340	17000	Fehler registrieren
16450	19000	Undefiniertes Symbol in Tabelle eintr.

Verzweigungen nach außen :

in Ze	nach	Bedingung	Bemerkung
16110	RETURN	LEN(T$) GT 4 AND LEN(T$) NE 8	Falsche Länge $Term
16530	RETURN		normales Ende

3.2.8 Weitere Unterprogramme

Wert in Tabelle der undefinierten Symbole eintragen

```
19000 REM * WERT IN DATEI DER UNDEF. SYMBOLE EINTRAGEN  *
19010 U=U+1
19020 UA(U)=UA
19030 UT(U)=UT
19040 UN$(U)=UN$
19050 RETURN
```

Die Variable U, die die Anzahl der undefinierten Symbole angibt, wird um eins erhöht. Das Feld UT() beinhaltet den Typ, das Feld UA() die Adresse und das Feld UN$() den Namen der undefinierten Symbole.

Symbol-Tabelle speichern und drucken

```
20000 REM * SYMBOLTABELLE SPEICHERN UND DRUCKEN          *
20010 PRINT#4
20020 PRINT#4,"SYMBOLE:"
20030 PRINT#4,"NAME   T WERT"
20040 PRINT#4,"------------"
20050 PRINT#15,"S:"+F$+".SYM"
20060 OPEN5,8,5,F$+".SYM,S,W"
20070 GOSUB25000
20080 IFDSTHENPRINTDS$:STOP
20090 GOSUB20700
20100 CLOSE5
20110 CMD4
20120 GOSUB20500
20130 PRINT#4
20140 RETURN
```

Wenn das Quellprogramm fertig assembliert ist, so ist auch die Symboltabelle komplett. Dieses kurze Unterprogramm gibt nun die gesamte Symboltabelle auf dem Drucker bzw. auf dem Bildschirm aus und speichert sie zusätzlich in einer sequentiellen Datei auf Floppy.

In Zeile 20060 wird dazu die Datei #5 auf der Floppy mit dem Dateinamen F$+".SYM" zum Schreiben geöffnet. Durch Aufruf der Unterprogramme ab Zeile 20500 und ab 20700 (vgl. Kapitel 3.2.1) werden alle Symbole nacheinander sowohl auf Floppy als auch auf dem Protokoll ausgegeben. Die Ausgabe wird mit dem CMD-Befehl auf die Protokoll-Datei umgelenkt, weil im Unterprogramm ab Zeile 20500 normale PRINT-Befehle verwendet werden.

Anschließend wird die Floppydatei geschlossen und das Unterprogramm verlassen.

Manuelle Eingabe von Symbolwerten

```
21000 REM * MANUELLE EINGABE VON SYMBOLWERTEN          *
21010 PRINTUN$;
21020 INPUTT$
21030 T$=LEFT$(T$+SP$,8)
21040 POKEQ4,0
21050 GOSUB16000
21060 IFPEEK(Q4)THEN21000
21070 IFUT=1THENTY=1:SI=1
21080 IFUT=2THENTY=1:SI=2
21090 IFUT=3ORUT=4THENTY=1:SI=1
21100 IFUT=5ORUT=6THENTY=1:SI=2
21110 IFUT=7ORUT=8THENTY=2:SI=2
21120 IFW>=256↑SITHEN21000
21130 TV=W
21140 TN$=UN$:TV=W:GOSUB18000
21150 RETURN
```

Wenn der Assembler beim nachträglichen Einsetzen der vorher undefinierten Symbole feststellt, daß ein bestimmtes Symbol immer noch nicht definiert ist, müßte eigentlich eine Fehlermeldung 'Symbol nicht definiert' ausgegeben werden. In dem vorliegenden Assembler wurde jedoch eine andere Möglichkeit gewählt. Ein Symbol das nicht definiert wurde, wird am Bildschirm erfragt. Dadurch erspart man sich den erneuten Lauf des Assemblers durch das gesamte Programm.

Zur Feststellung des Wertes wird vom Bildschirm die Zeichenreihe T$ eingelesen und sodann dem Unterprogramm 16000

übergeben, das den Wert eines Einzelausdruckes bestimmt. Dadurch kann die Eingabe auch hexadezimal oder binär erfolgen. War die Eingabe mit Fehlern behaftet, so wird nochmal nach dem Symbol gefragt. Der Typ des neuen Symbols (TY) richtet sich nach dem Typ des undefinierten Symbols (UT). In Zeile 21130 wird schließlich das neue Symbol in die Symboltabelle eingetragen und das Unterprogramm verlassen.

Eine Zeile einlesen

```
22000 REM * EINE ZEILE EINLESEN                                    *
22010 IFFT=1THENGOSUB22700:IS=ST:ZN=ZN+1:GOTO22050
22020 GOSUB22500
22030 IFPEEK(Q2)=0ANDPEEK(Q2+1)=0THENIS=64:RETURN
22040 ZN=PEEK(Q1)+256*PEEK(Q1+1)
22050 ZN$=RIGHT$("     "+STR$(ZN),5)
22060 IFIS>0ANDFP<>2THENFP=2:IS=AS
22070 RETURN
```

Dieses Unterprogramm liest eine Zeile der Quelldatei in die Variable T$ ein. Es wird unterschieden, ob die Eingabedatei eine sequentielle Datei oder eine Programmdatei ist. Diese Unterscheidung wird in der Variablen FT festgehalten. Bei FT=1 ist die Eingabedatei eine sequentielle Datei, und das Einlesen ist durch das Unterprogramm ab Zeile 22700 zu realisieren. In der Variablen IS wird der Status der Eingabedatei festgehalten. IS=64 zeigt das Ende der Quelldatei an.

Im Fall der sequentiellen Eingabedatei sind keine Zeilennummern im Quelltext vorhanden, so daß die Zeilen fortlaufend durchnumeriert werden. Wenn die Eingabedatei eine Programmdatei ist, so wird das Unterprogramm ab Zeile 22500 aufgerufen, und anschließend werden die von diesem Unterprogramm übergebenen Werte, die unter anderem auch die Programmzeile enthalten, ausgewertet. Wenn das Programm zu Ende ist (angezeigt durch PEEK(Q2+1)=0), so wird die Variable IS auf 64 gesetzt, um das Ende des Programms anzuzeigen.

Nachdem in Zeile 22050 die Variable ZN$ mit einer fünf-

stelligen Zeichenreihe, gebildet aus der Zeilennummer, besetzt wurde, wird das Unterprogramm verlassen.

Bestimmung des Wertes einer Binärzahl

```
23000 REM * BESTIMMUNG DES WERTES EINER BINAERZAHL       *
23010 BB=0
23020 FORI=1TOLEN(BB$)
23030 B=ASC(MID$(BB$,I))-48
23040 IFB<0ORB>1THENER=15:GOSUB17000:RETURN
23050 BB=2*BB+B
23060 NEXT
23070 RETURN
```

Dieses Unterprogramm legt den Wert einer Binärzahl, die als Zeichenreihe BB$ übergeben wird, in der Variablen BB ab. Die Berechnung des Wertes geschieht in einer Schleife, indem der Hilfsvariablen B zunächst der Wert einer einzelnen Stelle der Binärzahl zugeordnet wird. Die Variable BB erhält man nun, indem man jeweils den alten Wert der Variablen BB verdoppelt und den Wert der einzelnen Stelle hinzuaddiert. Man kann sich überlegen, daß durch dieses Verfahren - das den Mathematikern als Horner-Schema bekannt ist - am Ende der Schleife der richtige Wert der Variablen BB steht.

Nummer der Direktive bestimmen

```
24000 REM * NUMMER DER DIRECTIVE BESTIMMEN               *
24010 FORD=1TOD9
24020 A$=D$(D):GOSUB50
24040 IFA=0THENNEXT
24050 IFD>D9THEND=0
24060 RETURN
```

Dieses Unterprogramm durchsucht die Variable AA$ nach einer vorhandenen Direktive. Dazu wird jede mögliche Direktive mit Hilfe des Unterprogramms ab Zeile 50 untersucht und anschließend die Nummer der eventuell gefundenen Direktive in der Variablen D zurückgegeben.

Disk-Status-Werte DS und DS$ bestimmen

```
25000 REM * DISK-STATUS-WERTE DS UND DS$ BESTIMMEN       *
25010 INPUT#15,DS,D1$,D2$,D3$
25020 DS$=STR$(DS)+","+D1$+","+D2$+","+D3$
25030 RETURN
```

Hier wird der Status der Floppy aus dem Fehlerkanal in die Variablen DS und DS$ übertragen.

Die Bezeichnung der Ausgabevariablen wurde in Anlehnung an Basic 4.0 der Commodore 8000er Serie gewählt.

Fehlermeldung beim nachträglichen Einsetzen

```
26000 REM * FEHLERMELDUNG BEIM EINTRAGEN AUSGEBEN        *
26010 HH=AD:GOSUB250
26020 PRINT#4,"FEHLER BEI "HH$
26030 A=0
26040 RETURN
```

Beim nachträglichen Einsetzen von Symbolwerten in die Objektdatei können zwei Fehler auftauchen. Zum einen kann ein relativer Sprung, der ja nur 128 Byte umfassen kann, auf ein Sprungziel außerhalb dieses Bereichs zeigen, zum anderen kann der Wert eines 1-Byte-Operand größer als 255 sein. Wenn einer der beiden Fälle auftritt, wird das vorliegende Unterprogramm aufgerufen, das schließlich eine entsprechende Meldung auf dem Protokoll ausgibt und den Wert des Operanden mit dem Vorgabewert Null besetzt.

3.3 Hauptprogramm mit Vorspann

Vorspann

```
1 GOTO50000
50000 REM #  P R O G R A M M  -  V O R S P A N N        #
50170 Q1=49200:REM PZNR
50180 Q2=49202:REM PZVP
50190 Q3=49205:REM ZEICH
50200 Q4=49208:REM FEHLANZ
50210 GOSUB50500
50220 OPEN15,8,15
50230 DIMAC(5)
50240 FORI=0TO5
50250 READAC(I)
50260 NEXT
50280 DIMKM(58),KC$(58,10)
```

```
50290 FORI=1TO58
50300 READK$(I),KM(I):REM SPAETER NUR KM(I)
50310 IFAC(KM(I))=0THEN50330
50320 FORJ=0TOAC(KM(I))-1:READKC$(I,J):NEXTJ
50330 NEXTI
50340 D9=6
50350 DIMD$(D9)
50360 FORI=1TOD9
50370 READD$(I)
50380 NEXT
50385 FORI=128TO203:READBK$(I):NEXT:REM SIEHE 50600
50390 E9=24
50400 DIMER$(E9)
50410 FORI=1TOE9
50420 READER$(I)
50430 NEXT
50440 SP$="          "
50450 AN$=CHR$(34)
50460 DIMA(20),UA(200),UT(200),UN$(200)
50470 GOTO1000
```

In Zeile 1 des Programms steht ein Sprung auf Zeile 50000, denn aus Rechenzeitgründen ist es sinnvoll, den Vorspann eines Programms zeilennummernmäßig an den Schluß zu legen. In diesem Vorspann werden im wesentlichen die Felder mit den Werten aus den DATA-Statements besetzt, sowie einige globale Variablen mit konstanten Werten belegt. Anschliessend wird zu Zeile 1010 gesprungen, wo das eigentliche Hauptprogramm beginnt.

Assemblieren

```
1000 REM #  A S S E M B L I E R E N                        #
1010 FP=2
1020 INPUT"DATEI                  ";F$
1030 FT=2
1040 OPEN2,8,2,F$+".SRC,P"
1050 GOSUB25000
1060 IFDS=0THENGOSUB10:GOSUB10:GOTO1130
1070 IFDS<>64THENPRINTDS$:CLOSE2:GOTO1020
1080 CLOSE2
1090 FT=1
1100 OPEN2,8,2,F$+".SRC,S"
1110 GOSUB25000
1120 IFDSTHENPRINTDS$:CLOSE2:GOTO1020
1130 PRINT"[illegible]EINGABEDATEI IST "F$".SRC"
1140 PRINT#15,"S:"+F$+"...."
1150 PRINT#15,"S:"+F$+".UND"
```

```
1160 OPEN3,8,1,F$+"....,P,W"
1170 GOSUB25000
1180 IFDSTHENPRINTDS$:STOP
1190 INPUT"█LIST-GERAET (ADR.)    3███";LD
1200 PRINT"█LISTE ZU ";
1210 IFLD<8THENOPEN4,LD:PRINT"GERAET";LD:GOTO1260
1220 PRINT"DATEI "F$".LST AUF GERAETENR."LD
1230 OPEN4,LD,4,"@:"+F$+".LST,S,W"
1240 GOSUB25000
1250 IFDSTHENPRINTDS$:STOP
1260 PRINT#4,"*** COMMODORE 64   6502-ASSEMBLER ***     "
1270 PRINT#4,"     VERSION 1.5  (09.03.84)"
1280 PRINT#4,"ASSEMBLIEREN   VON "F$".SRC"
1290 PRINT#4,"OBJECT-DATEI   IST "F$".OBJ"
1300 PRINT#4,"SYMBOL-TABELLE IST "F$".SYM"
1310 PRINT#4
1320 PRINT#4,"ZEILE  ADR.  OBJ     * QUELLTEXT"
1330 PRINT#4
1340 GOSUB22000
1350 TT$=T$
1360 GOSUB10000
1370 HH=AD:GOSUB250
1380 PRINT#4,ZN$"  "HH$;"  "LEFT$(C$+"          ",7);U$" "TT$
1390 EB=PEEK(Q4)
1400 IFEBTHENFORI=1TOEB:PRINT#4,"FEHLER: ";ER$(PEEK(Q4+I)):NEXT
1410 POKEQ4,0
1420 EN=EN+EB
1430 IFC$=""THEN1480
1440 AD=AD+1
1450 HH$=LEFT$(C$,2):GOSUB400:PRINT#3,CHR$(HH);
1460 C$=MID$(C$,3)
1470 GOTO1430
1480 IFIS=0THEN1340
1490 IFIS=64THENPRINT#4,"█DATEIENDE ERREICHT.
1500 PRINT#4
1510 PRINT#4,EN"FEHLER."
1520 IFLD<>3THENPRINTEN"FEHLER."
1530 CLOSE3
1540 CLOSE2
```

Die Zeilen 1010 bis 1540 enthalten den kompletten Ablauf zum Assemblieren einer Datei, ausschließlich dem Einsetzen der bisher undefinierten Symbole in die Objekt-Datei, was ab Zeile 2000 durchgeführt wird.

In Zeile 1020 wird zunächst der Name der zu übersetzenden Datei in die Variable F$ vom Bildschirm eingelesen. Der Dateiname wird hier ohne den Zusatz '.SRC' angegeben. Die Quelldatei kann entweder eine sequentielle Datei oder eine

Programmdatei sein. Diese Datei benennt man, um Verwechslungen zu vermeiden, am besten mit dem Anhang '.SRC' (für Source-Code).

Die Art der Quelldatei wird vom Programm selbstständig erkannt und in der Variablen FT festgehalten. Dabei bedeutet FT=1, daß die Eingabedatei sequentiell ist und FT=2, daß eine Programmdatei als Eingabedatei vorliegt. Die Unterscheidung geschieht durch den Versuch, die Eingabedatei als Programmdatei zu öffnen. War die geöffnete Datei eine sequentielle Datei, so erkennt das Floppy-Betriebssystem dies und gibt den Fehlercode 64 zurück. Daraufhin versucht das Programm, die Datei als sequentielle Datei zu öffnen.

Eine Programmdatei enthält in den ersten beiden Bytes die Startadresse, wohin es geladen werden soll. Diese Startadresse ist jedoch hier unerheblich und muß deshalb überlesen werden, was in Zeile 1060 geschieht.

In Zeile 1160 wird als logische Datei #3 eine Zwischendatei mit dem Namen der Datei und angehängten vier Punkten eröffnet. In diese Datei wird die vorläufige Version der Objektdatei geschrieben. In Zeile 1190 kann das Ausgabegerät angewählt werden. Möglich sind dabei folgende Eingaben:

3 - Bildschirm
4 - Drucker
8 - Floppy
9 - Floppy

Wenn als Ausgabegerät eine Floppy spezifiert wurde, wird eine sequentielle Datei F$+".LST" angelegt, auf der genau das gespeichert wird, was sonst auf dem Drucker erscheint.

Die Zeilen 1260 bis 1330 bilden die Kopfzeilen des Protokolls. Anschließend werden in einer Schleife jeweils einzeln die Zeilen aus dem Quellprogramm eingelesen und in der Variablen TT$ gespeichert. Diese Zeilen werden dann sogleich assembliert, und das Ergebnis wird auf dem Protokoll ausgegeben. Dann werden noch die Fehlermeldungen ausgegeben, falls welche vorhanden waren. Die Anzahl der Fehlermeldungen pro Zeile wird sogleich auf 0 zurückgesetzt. In der Variablen C$ ist nach dem Assemblieren eine hexadezimale Befehlsfolge gespeichert. Diese wird dann als Gruppe von zwei Zeichen, die jeweils ein Byte bilden, in die vorläufige Objekt-Datei geschrieben, bis die Variable C$ die Länge 0 hat.

In Zeile 1510 wird zum Einlesen der nächste Zeile gesprungen, wenn IS gleich 0 ist, d.h. wenn das Ende der Datei

noch nicht erreicht ist.

War die Datei zu Ende, so wird eine entsprechende Meldung ausgegeben, sowie die Anzahl der Fehler auf der Listdatei und am Bildschirm ausgegeben und sowohl die Eingabedatei als auch die vorläufige Objektdatei geschlossen. Damit ist das Programmstück 'Assemblieren' beendet.

Werte nachträglich einsetzen

```
2000 REM #  W E R T E   NACHTRAEGLICH  EINSETZEN        #
2010 PRINT#15,"S:"+F$+".OBJ"
2020 GOSUB25000
2030 IFDS>1THENPRINTDS$:STOP
2040 IFUTHEN2090
2050 PRINT#15,"R:"+F$+".OBJ="+F$+"...."
2060 GOSUB25000
2070 IFDSTHENPRINTDS$:STOP
2080 GOTO2510
2090 OPEN3,8,1,F$+".OBJ,P,W"
2100 GOSUB25000
2110 IFDSTHENPRINTDS$:STOP
2120 OPEN2,8,2,F$+"....,P,R"
2130 GOSUB25000
2140 IFDSTHENPRINTDS$:STOP
2150 U1=0
2160 AD=SA-3:REM STARTADRESSE UEBERLESEN
2170 U1=1:UA=UA(U1):UT=UT(U1):UN$=UN$(U1)
2180 AD=AD+1
2190 GOSUB10:IS=ST:II=PEEK(Q3)
2200 IFUA>ADORU1>UTHENPRINT#3,CHR$(II);:GOTO2440
2210 SI=0
2220 IFUT=1ORUT=3ORUT=4ORUT=8THENSI=1
2230 IFUT=2ORUT=5ORUT=6ORUT=7THENSI=2
2240 IFSI=2THENI1=II:GOSUB10:IS=ST:I2=PEEK(Q3)
2250 IFUT=0THENSTOP
2260 T$=LEFT$(T$+SP$,8)
2270 TN$=UN$:GOSUB500
2280 IFTY=0THENGOSUB21000
2290 IFUT=8THEN2380
2300 A=II-158+TV
2310 IFSI=1AND(A<0ORA>255)THENGOSUB26000
2320 IFSI=1THENPRINT#3,CHR$(II-128+TV);:GOTO2420
2330 A=I1+256*I2-32768-128+TV
2340 H=INT(A/256)
2350 PRINT#3,CHR$(A-256*H)CHR$(H);
2360 AD=AD+1
2370 GOTO2420
```

```
2380 A=II+256*(II>127)+TV-(AD+1)
2390 IFA>127ORA<-128THENGOSUB26000
2400 IFA<0THENA=A+256
2410 PRINT#3,CHR$(A);
2420 IFU1=UTHENU1=U+1:GOTO2440
2430 U1=U1+1:UA=UA(U1):UT=UA(U1):UN$=UN$(U1)
2440 IFIS=0THEN2180
2450 CLOSE2
2460 CLOSE3
2470 CLOSE6
2480 PRINT#15,"S:"+F$+"...."
2490 GOSUB25000
2500 IFDS>1THENPRINTDS$:STOP
2510 PRINT#4,"FERTIG ASSEMBLIERT."
2520 IFLD<>3THENPRINT"FERTIG ASSEMBLIERT.."
2530 GOSUB20000
2540 CLOSE15
2550 CLOSE4
2560 END
```

In diesem Programmstück von Zeile 2010 bis 2560 wird aus der vorläufigen Objektdatei die endgültige Fassung gebildet, indem alle bisher undefinierten Symbole jeweils an die richtige Stelle eingetragen werden.

Dazu wird zunächst eine eventuell schon vorhandene Version der Objektdatei gelöscht. War kein Symbol undefiniert (U=0) wird einfach die vorläufige Objektdatei zur endgültigen Objektdatei umbenannt, was in Zeile 2050 geschieht. Im anderen Fall wird eine endgültige Objektdatei als File #3 eröffnet und die bisherige Objektdatei als File #2.

Es wird jeweils ein Zeichen aus File #2 gelesen und wieder in File #3 weggeschrieben, solange bis die in der Variablen AD mitgezählte Adresse kleiner oder gleich der Adresse des ersten undefinierten Symbols ist. Dann wird anstatt dem Wert in der vorläufigen Quelldatei der errechnete Wert in die endgültige Objektdatei geschrieben. Die Berechnung des endgültigen Wertes ist jedoch nicht ganz einfach. Insbesondere muß festgestellt werden, ob ein oder zwei Byte korrigiert werden müssen. Die Anzahl der zu korrigierenden Bytes wird in der Variablen SI festgehalten. Wenn der Typ (UT) des undefinierten Symbols gleich 1, 3, 4 oder 8 ist, muß ein Byte ersetzt werden, sonst zwei Byte.

In den Zeilen 2260 bis Zeilen 2280 wird der Wert des undefinierten Symbols festgestellt. Wenn der Name nicht in der Tabelle enthalten ist (TY=0) so wird der Wert mit Hilfe des Unterprogramms ab Zeile 21000 vom Bildschirm eingelesen.

Die Berechnung der einzusetzenden Werte muß unterschiedlich erfolgen, je nachdem, ob das undefinierte Symbol aus einem relativen Sprung resultiert (UT=8), oder ob ein oder zwei Byte korrigiert werden müssen.

Die Behandlung der relativen Sprünge geschieht in den Zeilen 2380 bis 2410, ein Byte wird in Zeile 2300 bis 2320 korrigiert und zwei Bytes in den Zeilen 2330 bis 2350.

Bei Zeile 2420 vereinigen sich die drei Zweige des Programms wieder. Dort wird die Variable U1, die auf das nächste zu ersetzende Symbol zeigt, um eins erhöht. Wenn die Datei noch nicht zu Ende ist (IS=0), wird zur Zeile 2180 gesprungen, wo das nächste Byte der vorläufigen Objektdatei gelesen wird. Wurde das Ende der Datei erreicht, so wird die Zwischendatei gelöscht und die Meldung 'Fertig assembliert.' ausgegeben. Danach wird noch die Symboltabelle auf Drucker und auf Floppy mit Hilfe des Unterprogramms ab Zeile 20000 ausgegeben.

Der Assembler hat also im Endeffekt zwei Dateien neu angelegt. Zum einen die endgültige Objektdatei und zum anderen eine sequentielle Datei, die die Symbole enthält. Diese Symboldatei kann von dem in Kapitel 5 beschriebenen Disassembler zur Wiedergewinnung von Symbolen genutzt werden.

Die Quelldatei wurde nur gelesen und deshalb nicht verändert.

Hauptprogramm mit Vorspann

Sprung auf Zeile 50000	1
Vorspann	50000 - 50470
Assemblieren	1030 - 1540
Werte nachträglich einsetzen	2000 - 2560

Variablen:

Name	Typ	Bereich	Bedeutung
A	H	-65535...65535	Hilfsvariable
AD	G	0...65535	Aktuelle Obj.-Adresse
C$	R	0..6 Zeichen	Assemblierter Code
D9	G	6	Anz. Direktiven

DS	R	0..99	Disk-Status (Nummer)
DS$	R	Zeichenreihe	DS,D1$,D2$,D3$ kombin.
E9	G	24	Anz. Fehlermeldungen
EB	H	0...15	Anz. Fehler pro Zeile
EN	G	Ganzzahlig	Anzahl Fehler insges.
F$	G	Zeichenreihe	Dateiname
FP	G	2,6	Log. Nr. Eingabedatei
FT	G	1,2	Typ der Eingabedatei
H	H	0...255	Hilfsvariable
HE$	G	0123456789ABCDEF	Hexadezimal-Ziffern
HH	P	0...65535	Wert v. 4-stell. Hex-Z
HH$	R	4 Zeichen	4-stellige Hex-Zahl
I	H	0...300	Laufvariable
I1	H	0...255	1. Eingelesener Code
I2	H	0...255	2. Eingelesener Code
II	R	0...255	Eingelesener Code
IS	R	0,2,64,320	Einlese-Status
J	H	0...10	Laufvariable
LD	G	0..15	Ausgabe-Geräte-Adresse
Q1	G	49200	Adresse Prg.Zeilennr.
Q2	G	49202	Adresse Prg.Vorw.zeig.
Q3	G	49205	Adresse eingel. Zeich.
Q4	G	49208	Adresse Anz. Fehler
SA	H	0...65535	absolute Startadresse
SI	H	0...2	Anz. Byte im Operand
SP$	G	8 Leerzeichen	z. Auffüll. v. Strings
T$	R/P	Zeichenreihe	Eingelesene Zeile
TT$	H	Zeichenreihe	Kopie von T$
TA	G	0...300	Anzahl Symbole in Tab.
TN$	E	Zeichenreihe	Gesuchtes Symbol
TV	E/A	0..65535	Wert des Symbols
TY	E/A	0,1,2	Typ des Symbols
U	G	0...300	Anz. undef. Symbole
U1	H	0...300	zeigt auf nächstes
			undefinierte Symbol
UA	E	Zeichenreihe	Adresse, wo und. Sym.
			einzusetzen ist
UN$	E	Zeichenreihe	Name des und. Symbols
UT	E	0...8	Typ des undef. Symbols
W	A	0...65535	Wert des Einzelterms
ZN$	R	5 Zeichen	Zeilennr. als String

Felder (Arrays):

Name	Dimen.	Typ	Bereich	Bedeutung
A	20	G	0...255	Positionen der

				Sonderzeichen
AC	5	H	0...10	Anzahl Modes je Befehlstyp
BK$	203	G	Zeichenreihen	Basic-Befehle
D$	D9	G	Zeichenreihen	Direktiven
EB	10	G	0...E9	Fehlernummern
ER$	E9	G	Zeichenreihen	Fehlermeldungen
K$	K9	G	je 3 Zeichen	Mnemonics
KM	K9	G	0...5	Befehlstyp
KC$	K9,10	G	2 Hex-Ziffern	Maschinencodes
TN$	255	G	je 8 Zeichen	Symbole
TV	255	G	0...65535	Werte d. Symbole
TY	255	G	0,1,2	Typen d. Symbole
UA	200	G	0...65535	Adressen der undef. Symbole
UN$	200	G	Zeichenreihen	Namen der undef. Symbole
UT	200	G	0...8	Typ undef. Symb.

Dateien :

#	Name	T	Bemerkung
2	F$+".SRC"	s/p	Quelldatei
3	F$+"...."	p	Vorläufige Objektdatei
4	F$+".LST"	s	Listdatei
3	F$+".OBJ"	p	Endgültige Objektdatei
15	Floppy		Kommado-/Fehlerkanal der Floppy

Unterprogrammaufrufe :

in	nach	Zweck
1050	25000	Disk-Status abfragen
1060	10	Ein Zeichen einlesen
1110	25000	Disk-Status abfragen
1170	25000	Disk-Status abfragen
1240	25000	Disk-Status abfragen
1340	22000	Eine Zeile einlesen
1360	10000	Eine Zeile assemblieren
1370	250	4-stellige Hex-Zahl bilden
1450	350	Wert von 2-stelliger Hex-Zahl bestimmen
2020	25000	Disk-Status abfragen
2060	25000	Disk-Status abfragen
2100	25000	Disk-Status abfragen

2130	25000	Disk-Status abfragen
2190	10	Ein Zeichen aus Zwischendatei einlesen
2240	10	Ein Zeichen aus Zwischendatei einlesen
2270	500	Symbol in Tabelle suchen
2280	21000	Symbol manuell erfassen
2310	26000	Fehlermeldung ausgeben
2390	26000	Fehlermeldung ausgeben
2490	25000	Disk-Status abfragen
2530	20000	Symboltabelle ausgeben
50210	50500	Initialisierungsroutine aufrufen

Verzweigungen nach außen :

in Ze	nach	Bedingung	Bemerkung
1180	STOP	DS NE 0	Disk-Fehler
1250	STOP	DS NE 0	Disk-Fehler
2030	STOP	DS GT 1	Disk-Fehler
2070	STOP	DS NE 0	Disk-Fehler
2110	STOP	DS NE 0	Disk-Fehler
2140	STOP	DS NE 0	Disk-Fehler
2250	STOP	UT = 0	darf eigentlich nicht auftauchen
2500	STOP	DS GT 1	Disk-Fehler
2560	END		Normales Ende

3.4 Mögliche Erweiterungen

Der vorliegende Assembler ist bereits recht vielseitig. Aber man kann jedes Programm noch verbessern. Es gibt im wesentlichen zwei Gruppen von möglichen Verbesserungen. Die eine Gruppe dient dazu, den Assembler noch schneller zu machen, die andere umfaßt neue Möglichkeiten wie zusätzliche Direktiven oder andere Möglichkeiten einen Operanden anzugeben.

Die in Kapitel 4 abgebildeten Maschinenprogramme verschnellern den Assembler etwa um den Faktor 10. Die Maschinenprogramme sind allgemeiner gehalten, so daß man trotzdem wirkungsvolle Änderungen im Basic-Programm durchführen kann, ohne daß man die Maschinenprogramme ändern muß. Wie man die Maschinenprogramme in den Assembler einbaut, entnehmen Sie bitte dem Listing im Anhang. Zur Erstellung der Maschinenprogramme können Sie den Basic-Assembler verwenden.

Natürlich können Sie weitere Basic-Unterprogramme in Assembler übertragen. Es bieten sich hier vor allem die Unterprogramme zum Auswerten von Ausdrücken an, weil hier viel Rechenzeit gespart werden kann, aber nicht zu viele globale Basic-Variablen verwendet werden, die man ja alle als Parameter an das Maschinenprogramm übergeben muß. Hilfreich wäre auch eine Routine, welche das Übertragen von Zeichen aus der vorläufigen Objekt-Datei in die endgültige unterstützt.

Neue Möglichkeiten können Sie durch Einbau zusätzlicher Direktiven gewinnen. Sinnvoll wäre z.B. eine Format-Direktive, mit der Sie das Protokoll auf dem Drucker mit definierter Anzahl von Zeichen pro Zeile und Zeilen pro Seite ausgeben können. Außer der %INCLUDE-Direktive können Sie auch Direktiven zum Einbinden von Symboltabellen oder fertigen Objekt-Dateien erstellen. Sie können auch leicht die Direktiven-Kennzeichnung "%" durch ein anderes Zeichen ersetzen, wenn Sie das "%"-Zeichen für andere Zwecke vorsehen wollen.

Ein Ausdruck könnte auch durch den ASCII-Code eines Zeichens gegeben werden, so daß man dann schreiben kann: LDA #"A" anstatt LDA #$41. Weiterhin können Sie versuchen, die Beschränkung von nur einem Verknüpfungszeichen im Operanden aufzuheben.

Sie sehen, es gibt vieles zu ergänzen. Wir haben in diesem Buch nicht alle Möglichkeiten realisiert, weil es nicht Sinn dieses Buches ist, nur das Listing eines perfekten Assemblers abzubilden, vielmehr sollte gezeigt werden, was ein Assembler macht, und wo die Probleme bei der Programmierung liegen.

4

Maschinenprogramme zum Assembler

4. Maschinenprogramme zum Assembler

Der im vorigen Kapitel vorgestellte Assembler ist zwar recht leistungsfähig und recht gut zu verstehen, da er ganz in Basic geschrieben ist, jedoch ist er noch sehr langsam, vor allem bei sehr langen Quelldateien. Deshalb wollen wir in diesem Kapitel einige Routinen vorstellen, die die häufig benötigten Basic-Unterprogramme und auch einige seltener benötigte durch Maschinenprogramme ersetzen, wodurch eine Geschwindigkeitserhöhung um den Faktor 10 erreicht wird (vgl. Unterprogramme in Kapitel 3.2.1).

Die vorgestellten Maschinenroutinen sind auch für andere Basic-Programme nützlich, z.B. auch für den Disassembler. Die Quelldatei für die gesamten Routinen wurde "ASS.SRC" genannt. Dementsprechend ist die Objektdatei, die zu Beginn des Programms zu laden ist "ASS.OBJ".

4.1 Konstanten-Vereinbarungen / ROM-Routinen

```
 10  C000         %ORG $C000
100  C000         ;******************************
110  C000         ;* KONSTANTENVEREINBARUNGEN *
120  C000         ;******************************
130  C000         ADRL=$14      ; INTEGERZAHL LOW-BYTE
140  C000         ADRH=ADRL+1   ;      ''      HIGH-BYTE
141  C000         ;ADRL,ADRH WERDEN VON DER ROUTINE
142  C000         ;GETADR BZW. GETAB BESETZT
143  C000         ;
150  C000         VARADL=$47 ; VARIABLEN-ADRESSE WIRD VON
151  C000         ;VARSUC IN VARADL,VARADL+1 ABGELEGT.
152  C000         ;
160  C000         STRADR=$1F ; ZELLENPAAR ZUR SPEICHERUNG
161  C000         ;DER STRING-ANFANGSADRESSE
162  C000         ;
165  C000         STRADR1=$22 ; ZELLENPAAR ZUR SPEICHERUNG
166  C000         ;DER STRING-ANFANGSADRESSE
167  C000         ;
170  C000         STRLEN=$21 ; ZELLE ZUR SPEICHERUNG
171  C000         ;DER STRING-LAENGE
172  C000         ;
175  C000         STRLEN1=$1E ; ZELLE ZUR SPEICHERUNG
176  C000         ;DER STRING-LAENGE
177  C000         ;
180  C000         BEP=$0200 ; BASIC-INPUT-PUFFER
181  C000         ;
```

```
190  C000       HOCHKM=$0F  ; FLAG FUER HOCHKOMMAMODUS
191  C000       ;
200  C000       AKTIO=$13  ; AKT. I/O-GERAET
201  C000       ;
210  C000       KTAB=$A09E; TABELLE MIT BASIC-KEYWORDS
211  C000       ;
220  C000       BASBZ=$7A  ; BASIC-BEFEHLS-ZEIGER
221  C000       ;
230  C000       WL=$62; HILFSZELLEN FUER
231  C000       WH=$63; DIVERSE ZWECKE
232  C000       ;
240  C000       NAMTAB=$9800 ;TABELLE DER SYMBOLNAMEN
250  C000       VALTAB=$9600 ;TABELLE DER SYMBOLWERTE
260  C000       TYPTAB=$9500 ;TABELLE DER SYMBOLTYPEN
270  C000       ;
280  C000       USRVEC=$0311; USR-VEKTOR
530  C000       GETBYT=$B79B ; LIES BYTE-WERT
531  C000       ;GETBYT LIEST BYTE-PARAMETER AUS
532  C000       ;BASIC-TEXT. WERT STEHT ANSCHLIESSEND
533  C000       ;IM X-REGISTER.
534  C000       ;
540  C000       FACADR=$B7F7 ; WANDELT FAC IN
541  C000       ;ADRESSFORMAT UM   NACH ADRL,ADRH
542  C000       ;
550  C000       GETAB=$B7EB  ; GETADR & GETBYT
551  C000       ;KOMBINIERTE ROUTINE:
552  C000       ;FRNNUM & FACADR & GETBYT
553  C000       ;
560  C000       FRMNUM=$AD8A ; LIES FLOAT.P.-WERT
561  C000       ;FRMNUM LIEST FLIESSKOMMA-PARAMETER AUS
562  C000       ;BASIC-TEXT. WERT STEHT ANSCHLIESSEND
563  C000       ;IM FLIESSKOMMA-AKKU (FAC).
564  C000       ;
570  C000       CHKCOM=$AEFD ; KOMMA UEBERLESEN
571  C000       ;CHKCOM UEBERLIEST EIN KOMMA IM BASICTEXT
572  C000       ;WENN KEIN KOMMA, DANN SYNTAX ERROR.
573  C000       ;
580  C000       FRMEVL=$AD9E ; HOLT AUSDRUCK AUS BASIC-TEXT
581  C000       ;NUMERISCHER AUSDRUCK STEHT DANN IM FAC
582  C000       ;STRING KANN MIT FRESTR GEHOLT WERDEN
583  C000       ;
590  C000       FRESTR=$B6A3 ; HOLT STRINGPARAMTER
591  C000       ;STRINGADRESSE IN $22,$23
592  C000       ;LAENGE STEHT IM AKKU
593  C000       ;
600  C000       FLDFACK=$BBA2; KONSTANTE NACH FAC
601  C000       ;FLDFACK LAEDT FAC MIT KONSTANTE.
602  C000       ;EINGABEPARAMETER:
603  C000       ; AKKU = LOW-BYTE KONSTANTENADRESSE
604  C000       ; X-REG= HIGH-BYTE KONSTANTENADRESSE
605  C000       ;
```

```
610   C000          FPLUSK=$B867 ; FAC = FAC+KONSTANTE
611   C000          ;PARAMETER SIEHE FLDFACK
612   C000          ;
640   C000          FACXY=$BBD4   ; FAC NACH ( X / Y )
641   C000          ;FACXY BRINGT FAC NACH VARIABLE
642   C000          ;EINGABEPARAMTER:
643   C000          ; X-REG = LOW -BYTE VARAIABLENADRESSE
644   C000          ; Y-REG = HIGH-BYTE VARAIABLENADRESSE
645   C000          ;
650   C000          FCHS=$BFB4    ; FAC = -FAC
651   C000          ; WECHSELT NUR DAS VORZEICHEN VON FAC
652   C000          ;
660   C000          FKMINUS=$B850; FAC = KONSTANTE-FAC
661   C000          ;PARAMETER SIEHE FLDFACK
662   C000          ;
670   C000          FMALK=$BA28   ; FAC = FAC*KONSTANTE
671   C000          ;PARAMETER SIEHE FLDFACK
672   C000          ;
680   C000          FKDIV=$BB0F   ; FAC = KONSTANTE/FAC
681   C000          ;PARAMETER SIEHE FLDFACK
682   C000          ;
690   C000          FSGN=$BC2B    ;  A = SGN(FAC)
691   C000          ;BRINGT VORZEICHEN VON FAC IN AKKU
692   C000          ;UND SETZT CARRY- UND ZERO-FLAG
693   C000          ;WENN FAC POSITIV,
694   C000          ;DANN AKKU=1 , CARRY=1 , ZERO=0
695   C000          ;WENN FAC = 0,
696   C000          ;DANN AKKU=0 , CARRY=1 , ZERO=1
697   C000          ;WENN FAC NEGATIV,
698   C000          ;DANN AKKU=$FF, CARRY=0 , ZERO=0
699   C000          ;
700   C000          FLPINT=$BC9B  ;FAC NACH INT
701   C000          ;BRINGT GANZZAHLIǴEN TEIL DES
702   C000          ;FAC NACH INTL,INTH .
703   C000          ;
710   C000          VARSUC=$B08B  ;SUCHT VARIABLE
711   C000          ;BZW. LEGT SIE NEU AN
712   C000          ;VARIABLENADRESSE IN $47,$48
713   C000          ;
730   C000          NEUSTR=$B4F4 ;NEUEN STRING
731   C000          ;EINRICHTEN
732   C000          ;
740   C000          ERROR=$A437 ;FEHLERMELDUNG
741   C000          ;AUSGEBEN (NUMMER=X-REG.)
742   C000          ;
750   C000          GETIN=$E124 ; HOLT ZEICHEN VON
751   C000          ;AKT. EINGABEFILE
752   C000          ;
760   C000          CHKIN=$E11E ; SETZT
761   C000          ;AKT. EINGABEFILE
762   C000          ;
765   C000          CHKOUT=$E118 ; SETZT
```

```
766   C000              ;AKT. AUSGABEFILE
767   C000              ;
770   C000              CLRCH=$FFCC ; SETZT
771   C000              ;FILE ZURUECK
772   C000              ;
775   C000              BSOUT=$E10C ; GIBT EIN ZEICHEN AUS
776   C000              ;
777   C000              STROUT=$AB25 ; GIBT STRING AUS
778   C000              ;LGE IN X-REG; ADR. IN $22,$23
779   C000              ;
780   C000              AYFLP=$B391 ; WANDELT 16-BIT-ZAHL
781   C000              ;IN A/Y NACH FLIESSKOMMA
782   C000              ;
790   C000              YFLP=$B3A2 ; WANDELT 8-BIT-ZAHL
791   C000              ;IN Y NACH FLIESSKOMMA
792   C000              ;
800   C000              ILLQERR=$B248; 'ILLEGAL QUANTITY'
810   C000              SPOUT=$AB3F ;LEERZEICHEN AUSGEBEN
```

Dieser Abschnitt der Quelldatei vereinbart Konstanten, das sind meistens Adressen in der Zero-Page, die zur Zwischenspeicherung und zur Indizierung verwendet werden. Außerdem werden die ROM-Routinen vereinbart, deren Bedeutung Sie Kapitel 2.4 entnehmen können.

4.2 Sprungtabelle und Hilfszellen

```
1000  C000              ;******************************
1010  C000              ;*  SPRUNGTABELLE             *
1020  C000              ;******************************
1025  C000              ASS:
1040  C000  4C8080 R JMP BLANKELI
1060  C003  4C8080 R JMP GETZEI
1080  C006  4C8080 R JMP GETPZ
1100  C009  4C8080 R JMP GETTZ
1120  C00C  4C8080 R JMP HEX4S
1140  C00F  4C8080 R JMP HEX2S
1160  C012  4C8080 R JMP HEXDEZ
1180  C015  4C8080 R JMP INDEX
1200  C018  4C8080 R JMP SONDZ
1220  C01B  4C8080 R JMP FEHLREG
1240  C01E  4C8080 R JMP TABSUCH
1260  C021  4C8080 R JMP TABEINF
1280  C024  4C8080 R JMP MNEMO
1300  C027  4C8080 R JMP TABDRUCK
1320  C02A  4C8080 R JMP TABSPEI
1340  C02D  4C8080 R JMP INIT
```

```
1500  C030        ;*****************************
1510  C030        ;* VARIABLE UND HILFSZELLEN *
1520  C030        ;*****************************
1530  C030  0000  PZNR:WOR 0 ; ZEILENNUMMER
1531  C032        ;DER PROGRAMMZEILE
1540  C032  0000  PZVP:WOR 0 ; VORWAERTS-
1541  C034        ;ZEIGER DER PROGRAMMZEILE
1542  C034        ;
1550  C034  00    ZLEN: BYT 0 ; ZUM ZWISCHEN-
1551  C035        ;SPEICHERN DER ZEILENLAENGE
1552  C035        ;
1560  C035  00    ZEICH: BYT 0 ; VON GETZEI
1561  C036        ;EINGELESENES ZEICHEN
1562  C036        ;
1570  C036  00    ANZ: BYT 0 ; ANZAHL DURCHLAEUFE
1571  C037        ;BEI INDEX-FUNKTION
1572  C037        ;
1580  C037  00    WERT: BYT 0 ;U.A. FUER HEX-ROUTINE
1581  C038        ;
1590  C038  00    FEHLANZ: BYT 0
1600  C039        FEHLERT:
1601  C048  00    %DUP 16,BYT 0
1602  C049        ;
1610  C049  0000  SYMANZ: WOR 0 ;ANZAHL SYMBOLE
1611  C04B        ;
1620  C04B  00    SYMTYP:BYT 0
1630  C04C  0000  SYMWERT:WOR 0
1640  C04E  00    SYMNR:BYT 0
1641  C04F        ;
1800  C04F  91    F65536:BYT $91
1810  C050  0000  WOR 0
1820  C052  0000  WOR 0
1830  C054        F65536H = F65536 / 256
1840  C054        H=F65536H * 256
1850  C054        F65536L = F65536 - H
```

Am Beginn des Programms steht eine Sprungtabelle, die auf die einzelnen Routinen verzweigt. Das hat, ähnlich wie bei der KERNAL-Sprungtabelle, den Sinn, daß neuere Versionen der Unterprogramme erstellt werden können, ohne daß sich die Aufrufadressen ändern.

Im Anschluß an die Sprungtabelle werden einige Hilfszellen deklariert, die zur Parameterübergabe oder zur Zwischenspeicherung von Werten benötigt werden. Außerdem wird die Fließkomma-Konstante 65536 definiert.

4.3 Leerzeichen eliminieren

```
2000  C054              ;****************************
2010  C054              ;*  BLANKS ELIMINIEREN      *
2020  C054              ;****************************
2030  C054              BLANKELI:
2040  C054  20FDAE      JSR CHKCOM
2050  C057  208BB0      JSR VARSUC
2060  C05A  208080 R    JSR STRPARL
2070  C05D  F000   R    BEQ BLANKEND
2080  C05F              BLANK1:
2090  C05F  B11F        LDA (STRADR),Y
2100  C061  C920        CMP #$20
2110  C063  D000   R    BNE BLANK3
2120  C065  E61F        INC STRADR
2130  C067  D000   R    BNE BLANK2
2140  C069  E620        INC STRADR+1
2150  C06B              BLANK2:
2160  C06B  C621        DEC STRLEN
2170  C06D  F000   R    BEQ BLANK9
2180  C06F  D0EE        BNE BLANK1
2190  C071              BLANK3:
2200  C071  A421        LDY STRLEN
2210  C073              BLANK4:
2220  C073  88          DEY
2230  C074  B11F        LDA (STRADR),Y
2240  C076  C920        CMP #$20
2250  C078  D000   R    BNE BLANK9
2260  C07A  C621        DEC STRLEN
2270  C07C  D0F5        BNE BLANK4
2280  C07E              BLANK9:
2290  C07E  208080 R    JSR STRPARS
2300  C081              BLANKEND:
2310  C081  60          RTS
2320  C082              ;
2400  C082              ;** STRINGPARAMETER LESEN **
2410  C082              STRPARL:
2420  C082  A002        LDY #2
2430  C084  B147        LDA (VARADL),Y
2440  C086  8520        STA STRADR+1
2450  C088  88          DEY
2460  C089  B147        LDA (VARADL),Y
2470  C08B  851F        STA STRADR
2480  C08D  88          DEY
2490  C08E  B147        LDA (VARADL),Y
2500  C090  8521        STA STRLEN
2510  C092  60          RTS
2520  C093              ;
2600  C093              ;** STRINGPARAMETER SCHREIBEN **
2610  C093              STRPARS:
```

```
2620  C093  A002      LDY #2
2630  C095  A520      LDA STRADR+1
2640  C097  9147      STA (VARADL),Y
2650  C099  88        DEY
2660  C09A  A51F      LDA STRADR
2670  C09C  9147      STA (VARADL),Y
2680  C09E  88        DEY
2690  C09F  A521      LDA STRLEN
2700  C0A1  9147      STA (VARADL),Y
2710  C0A3  60        RTS
2720  C0A4            ;
2800  C0A4            ;** STRINGPARAMETER LESEN **
2810  C0A4            STRPARL1:
2820  C0A4  A002      LDY #2
2830  C0A6  B147      LDA (VARADL),Y
2840  C0A8  8523      STA STRADR1+1
2850  C0AA  88        DEY
2860  C0AB  B147      LDA (VARADL),Y
2870  C0AD  8522      STA STRADR1
2880  C0AF  88        DEY
2890  C0B0  B147      LDA (VARADL),Y
2900  C0B2  851E      STA STRLEN1
2910  C0B4  60        RTS
2920  C0B5            ;
```

Wir wollen zunächst die drei Unterprogramme zum String-Parameter lesen und schreiben behandeln, die am Schluß dieses Programmstücks stehen. Diese Unterprogramme werden auch noch von anderen Routinen benötigt.

Wie wir in Kapitel 2.3 gesehen haben, werden String-Variable als drei Byte abgespeichert, das erste Byte gibt die Länge des Strings an, die beiden folgenden sind Low- und High-Byte der String-Anfangsadresse. Die drei abgebildeten Unterprogramme gehen davon aus, daß die Variablen-Adresse in den Zellen VARADL und VARADL+1 gespeichert ist.

Das Unterprogramm STRPARL liest aus der Variablen die String-Parameter und legt sie in die Speicherzellen STRLEN, STRADR und STRADR+1 ab. Danach kann auf die Länge der Strings und die Startadresse leichter zugegriffen werden. Das Unterprogramm STRPARS vollführt die umgekehrte Funktion, es überträgt die Werte aus den Zellen STRLEN, STRADR und STRADR+1 in die Variablenzellen.

Die Routine STRPARL1 macht das gleiche wie STRPARL, jedoch werden die Werte in den Zellen STRLEN1, STRADR1 und STRADR1+1 abgelegt.

Nun zum eigentlichen Unterprogramm BLANKELI. Am Anfang sehen Sie eine typische Möglichkeit, einen String-Variablen-Parameter einzulesen. Zunächst wird geprüft, ob im Basic-Text ein Komma folgt, dann wird die Variable gesucht und anschließend die String-Parameter eingelesen. Da die Routine STRPARL als letztes die Stringlänge liest, kann mit Sprung BEQ BLANKEND der Fall behandelt werden, daß die Zeichenreihe leer ist. Das Y-Register wurde im Unterprogramm STRPARL auf Null gesetzt, so daß sofort das erste Zeichen des Strings angesprochen werden kann. Ist es ein Leerzeichen, so wird einfach die String-Anfangsadresse um eins erhöht und die Stringlänge vermindert. Dieser Vorgang wird fortgesetzt, bis entweder die Stringlänge Null ist, oder ein anderes Zeichen als das Leerzeichen gefunden wird. In diesem Fall wird das letzte Zeichen der Zeichenreihe überprüft. Die Stringlänge wird solange vermindert, bis das letzte Zeichen des Strings kein Leerzeichen mehr ist. Dann werden die String-Parameter abgespeichert und das Unterprogramm verlassen.

4.4 Ein Zeichen einlesen

```
3000  C0B5             ;******************************
3010  C0B5             ;* EIN ZEICHEN EINLESEN       *
3020  C0B5             ;******************************
3030  C0B5             GETZEI:
3040  C0B5  209BB7     JSR GETBYT;FILE-NUMMER IM X-REGISTER
3050  C0B8  8613       STX AKTIO
3060  C0BA  201EE1     JSR CHKIN
3070  C0BD  2024E1     JSR GETIN
3080  C0C0  8D35C0     STA ZEICH
3090  C0C3  A613       LDX AKTIO
3100  C0C5  F000     R BEQ GETZEI1
3110  C0C7  20CCFF     JSR CLRCH
3120  C0CA  A200       LDX #0
3130  C0CC  8613       STX AKTIO
3140  C0CE             GETZEI1:
3150  C0CE  60         RTS
3160  C0CF             ;
```

Dieses Unterprogramm ersetzt den GET-Befehl. Der ASCII-Code des eingelesenen Zeichens steht anschließend in der Hilfszelle ZEICH. Hier wird zunächst aus dem Basic-Text die logische Filenummer der zu lesenden Datei in das X-Register gelesen. Anschließend wird der Kanal geöffnet, ein Zeichen geholt und der Kanal wieder geschlossen.

4.5 Eine Programmzeile lesen

In diesem Programmabschnitt sind drei Unterprogramme enthalten, das erste für die Zuweisung einer Zeichenreihe im Inputpuffer an eine String-Variable, das zweite zum Umwandeln eines Basic-Befehlscodes in Klartext und das dritte zum eigentlichen Einlesen einer Programmzeile aus einer Datei.

String-Zuweisung

```
3200  C0CF             ;** STRING-ZUWEISUNG **
3210  C0CF             STRZUW:
3220  C0CF  AD34C0     LDA ZLEN
3230  C0D2  C521       CMP STRLEN
3240  C0D4  F000     R BEQ STRZU1
3250  C0D6  9000     R BCC STRZU1
3260  C0D8  20F4B4     JSR NEUSTR
3270  C0DB  861F       STX STRADR
3280  C0DD  8420       STY STRADR+1
3290  C0DF             STRZU1:
3300  C0DF  8521       STA STRLEN
3310  C0E1  A8         TAY
3320  C0E2  F000     R BEQ STRZU3
3330  C0E4             STRZU2:
3340  C0E4  88         DEY
3350  C0E5  B90002     LDA BEP,Y
3360  C0E8  911F       STA (STRADR),Y
3370  C0EA  C000       CPY #0
3380  C0EC  D0F6       BNE STRZU2
3390  C0EE             STRZU3:
3400  C0EE  2093C0     JSR STRPARS
3410  C0F1  60         RTS
3420  C0F2             ;
```

Dieses kleine Unterprogramm weist den Inhalt des Input-Puffers (Zellen $0200 bis $024F) mit der Zeichenreihenlänge ZLEN einer String-Variablen zu. Dabei können zwei Fälle auftreten. Einmal kann die neue Zeichenreihe länger sein als die alte, dann muß ein neuer String eingerichtet werden, andernfalls kann der alte Stringbereich benützt werden. Das Einrichten eines neuen Strings geschieht mit der Basic-Routine NEUSTR, die die String-Adresse im X- und Y-Register übergibt.

Auf alle Fälle muß der Inhalt des Basic-Input-Puffers in den String-Bereich übertragen werden. Dies geschieht in

einer Schleife zwischen den Labeln STRZU2 und STRZU3. Anschließend werden auch die Sting-Parameter in die Variable übertragen und das Unterprogramm beendet.

Basic-Code umwandeln

```
3500  C0F2              ;*****************************
3510  C0F2              ;* BASIC-CODE UMWANDELN      *
3520  C0F2              ;*****************************
3530  C0F2              PZCODE:
3540  C0F2  AA          TAX
3550  C0F3  1000     R  BPL PZCODE6
3560  C0F5  C9FF        CMP #$FF ; CODE FUER PI
3570  C0F7  F000     R  BEQ PZCODE6
3580  C0F9  240F        BIT HOCHKM
3590  C0FB  3000     R  BMI PZCODE6
3600  C0FD  38          SEC
3610  C0FE  E97F        SBC #$7F
3620  C100  AA          TAX
3630  C101  A0FF        LDY #$FF
3640  C103              PZCODE2:
3650  C103  CA          DEX
3660  C104  F000     R  BEQ PZCODE4
3670  C106              PZCODE3:
3680  C106  C8          INY
3690  C107  B99EA0      LDA KTAB,Y
3700  C10A  10FA        BPL PZCODE3
3710  C10C  30F5        BMI PZCODE2
3720  C10E              PZCODE4:
3730  C10E  C8          INY
3740  C10F  B99EA0      LDA KTAB,Y
3750  C112  3000     R  BMI PZCODE5
3760  C114  AE34C0      LDX ZLEN
3770  C117  EE34C0      INC ZLEN
3780  C11A  9D0002      STA BEP,X
3790  C11D  D0EF        BNE PZCODE4
3800  C11F              PZCODE5:
3810  C11F  297F        AND #$7F
3820  C121              PZCODE6:
3830  C121  C922        CMP #$22 ; HOCHKOMMA
3840  C123  D000     R  BNE PZCODE7
3850  C125  A50F        LDA HOCHKM
3860  C127  49FF        EOR #$FF
3870  C129  850F        STA HOCHKM
3880  C12B  A922        LDA #$22
3890  C12D              PZCODE7:
3900  C12D  60          RTS
3910  C12E              ;
```

Diese Routine benötigt als Eingabeparameter den Akkumulator, der das zu dekodierende Zeichen enthält, und eine Zelle ZLEN, die angibt, ab welcher Stelle im Input-Puffer das dekodierte Ergebnis abgelegt werden soll. Die hier mit KTAB gezeichnete Tabelle ist die im ROM eingebaute Befehlstabelle des Interpreters. Hier sind die Befehle jeweils so gespeichert, daß beim letzten Buchstaben des Befehls das Bit 7 gesetzt ist.

Außerdem wird noch überprüft, ob der Hochkomma-Modus aktiv ist oder nicht. Bei eingeschaltetem Hochkomma-Modus braucht keine Codewandlung zu erfolgen.

Eine Programmzeile aus Datei einlesen

```
4000  C12E            ;******************************
4010  C12E            ;* EINE PROGRAMMZEILE LESEN *
4020  C12E            ;******************************
4030  C12E            GETPZ:
4040  C12E  A900      LDA #0
4050  C130  850F      STA HOCHKM
4060  C132  8D34C0    STA ZLEN ; EINGABELAENGE
4070  C135  209BB7    JSR GETBYT;FILE-NUMMER IM X-REGISTER
4080  C138  8613      STX AKTIO
4090  C13A  20FDAE    JSR CHKCOM
4100  C13D  208BB0    JSR VARSUC
4110  C140  2082C0    JSR STRPARL
4120  C143  A613      LDX AKTIO
4130  C145  201EE1    JSR CHKIN
4140  C148  2024E1    JSR GETIN
4150  C14B  8D32C0    STA PZVP
4160  C14E  2024E1    JSR GETIN
4170  C151  8D33C0    STA PZVP+1
4180  C154  A8        TAY
4190  C155  F000    R BEQ GETPZ2
4200  C157  2024E1    JSR GETIN
4210  C15A  8D30C0    STA PZNR
4220  C15D  2024E1    JSR GETIN
4230  C160  8D31C0    STA PZNR+1
4240  C163            GETPZ1:
4250  C163  2024E1    JSR GETIN
4260  C166  20F2C0    JSR PZCODE
4270  C169  AC34C0    LDY ZLEN
4280  C16C  990002    STA BEP,Y
4290  C16F  A8        TAY
4300  C170  F000    R BEQ GETPZ2
4310  C172  EE34C0    INC ZLEN
4320  C175  D0EC      BNE GETPZ1
4330  C177  A217      LDX #23 ;STRING TOO LONG
```

```
4340  C179  4C37A4     JMP ERROR
4350  C17C             GETPZ2:
4360  C17C  20CFC0     JSR STRZUW
4370  C17F  A613       LDX AKTIO
4380  C181  F000     R BEQ GETPZE
4390  C183  20CCFF     JSR CLRCH
4400  C186  A200       LDX #0
4410  C188  8613       STX AKTIO
4420  C18A             GETPZE:
4430  C18A  60         RTS
4440  C18B             ;
```

Zum Verständnis dieser Routine müssen wir zunächst untersuchen, wie eine Basic-Programmzeile im Speicher abgelegt ist. Zunächst erscheinen zwei Byte, die den Beginn der folgenden Programmzeile anzeigen. Sind diese beiden Byte gleich 0, so ist dadurch das Ende des Programms gekennzeichnet. Nach diesen zwei Byte folgen weitere zwei Byte, die die Zeilennummer im Basic-Text darstellen. Anschließend folgen bis zu 80 Zeichen, die jeweils noch dekodiert werden müssen, um dem Basic-Klartext zu entsprechen. Das Ende einer Zeile wird mit einem 0-Byte gekennzeichnet.

Das vorliegende Unterprogramm annulliert zunächst den Hochkomma-Modus, holt dann die String-Variable und eröffnet den Eingabekanal für die Datei.

Dann werden zwei Zeichen aus der Datei gelesen und in die Zellen PZVP und PZVP+1 gespeichert. Wenn der Inhalt der letzten Zelle gleich Null ist, so bedeutet dies ein Programmende, und es wird zum Label GETPZ2 verzweigt. Im Normalfall aber wird die Zeilennummer eingelesen und in die Zellen PZNR und PZNR+1 gespeichert. Dann wird in einer Schleife ein Zeichen gelesen, dekodiert und in den Basic-Input-Puffer abgelegt. Es ist die Ausgabe einer Fehlermeldung vorgesehen, wenn die Zeichenreihe länger als 255 wird. Wurde die Ende-Marke der Basic-Zeile, hier ein 0-Byte, gefunden, so wird zum Label GETPZ2 gesprungen, welches die Zuweisung an die String-Variable vornimmt und den Eingabekanal schließt.

4.6 Eine Textzeile lesen

```
4500  C18B             ;******************************
4510  C18B             ;* EINE TEXT-ZEILE EINLESEN *
4520  C18B             ;******************************
4530  C18B             GETTZ:
4540  C18B  A900       LDA #0
```

```
4550  C18D  8D34C0    STA ZLEN ; EINGABELAENGE
4560  C190  209BB7    JSR GETBYT;FILE-NUMMER IM X-REGISTER
4570  C193  8613      STX AKTIO
4580  C195  20FDAE    JSR CHKCOM
4590  C198  208BB0    JSR VARSUC
4600  C19B  2082C0    JSR STRPARL
4610  C19E  A613      LDX AKTIO
4620  C1A0  201EE1    JSR CHKIN
4630  C1A3            GETTZ1:
4640  C1A3  2024E1    JSR GETIN
4650  C1A6  AC34C0    LDY ZLEN
4660  C1A9  990002    STA BEP,Y
4670  C1AC  C90D      CMP #$0D ; CR PRINT
4680  C1AE  F0CC      BEQ GETPZ2 ;WEITER BEI GETPZ
4690  C1B0  EE34C0    INC ZLEN
4700  C1B3  D0EE      BNE GETTZ1
4710  C1B5  A217      LDX #23 ;STRING TOO LONG
4720  C1B7  4C37A4    JMP ERROR
```

Dieses Programm ist sehr ähnlich zum vorher beschriebenen 'Programmzeile lesen', jedoch entfällt hier das Einlesen des Vorwärtszeigers und der Zeilennummer. Das Endekriterium ist hier das Auftreten eines Carriage-Return ($OD).

Da die Stringzuweisung die gleiche ist, wie bei 'Programmzeile lesen', wird einfach beim Erreichen des Endekriteriums zum Label GETPZ2 gesprungen, das weiter oben beschrieben ist.

4.7 Wert von Hexadezimalzahl bestimmen

```
5000  C1BA              ;******************************
5010  C1BA              ;* WERT VON HEX-ZAHL          *
5020  C1BA              ;******************************
5030  C1BA              HEXDEZ:
5040  C1BA  2082B7      JSR $B782
5050  C1BD  8C34C0      STY ZLEN
5060  C1C0  C005        CPY #5
5070  C1C2  B000     R  BCS ILLHEX
5080  C1C4  A000        LDY #0
5090  C1C6  8463        STY WH
5100  C1C8  8462        STY WL
5110  C1CA              HEXDEZ1:
5120  C1CA  CC34C0      CPY ZLEN
5130  C1CD  B000     R  BCS HEXDEZ8
5140  C1CF  0662        ASL WL
5150  C1D1  2663        ROL WH
5160  C1D3  0662        ASL WL
5170  C1D5  2663        ROL WH
```

```
5180  C1D7  0662        ASL WL
5190  C1D9  2663        ROL WH
5200  C1DB  0662        ASL WL
5210  C1DD  2663        ROL WH
5220  C1DF  B122        LDA (STRADR1),Y
5230  C1E1  38          SEC
5240  C1E2  E930        SBC #$30
5250  C1E4  C90A        CMP #10
5260  C1E6  9000     R  BCC HEXDEZ2
5270  C1E8  E907        SBC #7
5280  C1EA  C910        CMP #16
5290  C1EC  B000     R  BCS ILLHEX
5300  C1EE              HEXDEZ2:
5310  C1EE  C8          INY
5320  C1EF  6562        ADC WL
5330  C1F1  8562        STA WL
5340  C1F3  90D5        BCC HEXDEZ1
5350  C1F5              ;
5360  C1F5              ILLHEX:
5370  C1F5  A215        LDX #21  ; ILL.HEXZAHL
5375  C1F7  208080   R  JSR FEHLER
5380  C1FA  A000        LDY #0
5390  C1FC  20A2B3      JSR YFLP
5400  C1FF  60          RTS
5410  C200              ;
5420  C200              HEXDEZ8:
5430  C200  A462        LDY WL
5440  C202  A563        LDA WH
5450  C204  3000     R  BMI HEXDEZ9
5460  C206  2091B3      JSR AYFLP
5470  C209  60          RTS
5480  C20A              ;
5490  C20A              HEXDEZ9:
5500  C20A  2091B3      JSR AYFLP
5510  C20D  A94F        LDA #F65536L
5520  C20F  A0C0        LDY #F65536H
5530  C211  2067B8      JSR FPLUSK
5540  C214  60          RTS
5550  C215              ;
```

Dieses Unterprogramm wird von Basic als USR-Funktion aufgerufen. Das Setzen des USR-Vektors wird im Basic-Programm erledigt (z.B. durch Aufruf der unten beschriebenen Routine INIT). Hier wird als Parameter für die Funktion eine String-Variable verwendet. Das Einlesen der String-Variable wurde bereits durch den Basic-Interpreter bewerkstelligt. Wir müssen hier allerdings noch mit Hilfe der Routine ab $B782 die String-Parameter übernehmen, dabei werden die String-Adresse in die Zellen $22 und $23 abgelegt, die wir hier mit STRADR1 und STRADR1+1 bezeichnet haben. Die Länge der Strings wird im Y-Register übergeben.

Wir speichern die Stringlänge noch zusätzlich in der Zelle ZLEN ab. Ist die Länge größer gleich fünf, so wird zur Behandlung eines Fehlers verzweigt. Der Wert der Hexadezimalzahl wird in den Hilfszellen WH und WL aufgebaut. Die ASL/ROL-Kombinationen stellen eine Multiplikation des aufgebauten Wertes mit 16 dar. Dazu wird jeweils der Wert des neuen Zeichens hinzuaddiert. Die Zuweisung der erhaltenen Werte an den Fließkomma-Akkumulator geschieht ab dem Label HEXDEZ8. Hier wird die Routine AYFLP verwendet, die eine im Akku und Y-Register stehende Integer-Zahl in Fließkomma umwandelt. Diese Routine interpretiert die Zahl jedoch als vorzeichenbehaftete 16-Bit-Integerzahl, so daß eine Korrektur vorgenommen werden muß (hier ab Label HEXDEZ9), die noch 65536 zum Ergebnis hinzuzählt, wenn die umzuwandelnde Zahl größer als 32767 war.

4.8 Hexadezimal-Zahl bilden

```
6000  C215           ;******************************
6010  C215           ;* HEX-ZIFFERN BILDEN         *
6020  C215           ;******************************
6100  C215           HEXTAB:
6110  C222  444546   %ASC "0123456789ABCDEF"
6120  C225           ;
6230  C225           HEX:
6240  C225  AD37C0   LDA WERT
6250  C228  290F     AND #$0F
6260  C22A  AA       TAX
6270  C22B  BD15C2   LDA HEXTAB,X
6280  C22E  8D0102   STA BEP+1
6290  C231  AD37C0   LDA WERT
6300  C234  4A       LSR A
6310  C235  4A       LSR A
6320  C236  4A       LSR A
6330  C237  4A       LSR A
6340  C238  AA       TAX
6350  C239  BD15C2   LDA HEXTAB,X
```

```
6360  C23C  8D0002   STA BEF
6370  C23F  60       RTS
6380  C240
6400  C240           HEX2S:
6410  C240  209BB7   JSR GETBYT
6420  C243  8E37C0   STX WERT
6430  C246  20FDAE   JSR CHKCOM
6440  C249  208BB0   JSR VARSUC
6450  C24C  2082C0   JSR STRPARL
6460  C24F  2025C2   JSR HEX
6470  C252  A902     LDA #2
6480  C254  8D34C0   STA ZLEN
6490  C257  20CFC0   JSR STRZUW
6500  C25A  60       RTS
6510  C25B           ;
6600  C25B           HEX4S:
6610  C25B  20FDAE   JSR CHKCOM
6620  C25E  208AAD   JSR FRMNUM
6625  C261  20F7B7   JSR FACADR
6630  C264  A514     LDA ADRL
6640  C266  8D37C0   STA WERT
6650  C269  A515     LDA ADRH
6660  C26B  8D36C0   STA ANZ
6670  C26E  20FDAE   JSR CHKCOM
6680  C271  208BB0   JSR VARSUC
6690  C274  2082C0   JSR STRPARL
6700  C277  2025C2   JSR HEX
6710  C27A  8D0202   STA BEF+2
6720  C27D  AD0102   LDA BEF+1
6730  C280  8D0302   STA BEF+3
6740  C283  AD36C0   LDA ANZ
6750  C286  8D37C0   STA WERT
6760  C289  2025C2   JSR HEX
6770  C28C  A904     LDA #4
6780  C28E  8D34C0   STA ZLEN
6790  C291  20CFC0   JSR STRZUW
6800  C294  60       RTS
6810  C295           ;
```

Zunächst befindet sich in diesem Programmstück eine Tabelle mit dem Label HEXTAB, welche die ASCII-Codes der hexadezimalen Ziffern enthält. Dann folgt ein Unterprogramm, welches eine zweistellige Hexadezimalzahl an den Anfang des Basic-Input-Puffers legt. Als Eingabeparameter dient die Zelle WERT.

Danach ist das Unterprogramm HEX2S abgebildet, das einer Basic-String-Variablen eine zweistellige hexadezimale Zei-

chenreihe zuweist. Zunächst wird der umzuwandelnde Wert gelesen und in der Zelle WERT gespeichert, dann wird die Variable gelesen und der Wert umgewandelt. Schließlich wird mit Hilfe der Routine STRZUW, die oben beschrieben ist, das Ergebnis an die Variable zugewiesen.

Ab dem Label HEX4S steht das Unterprogramm zum Bilden einer vierstelligen Hexadezimalzahl. Das Einlesen des Wertes geschieht hier mit den Aufrufen von FRMNUM und FACADR. Danach steht der Wert als Integerzahl in den Zellen ADRL und ADRH. Die Umwandlung in hexadezimale Ziffern geschieht durch zweimaligen Aufruf der Routine HEX.

4.9 INDEX-Funktion

```
7000  C295            ;******************************
7010  C295            ;*  POSITION SUCHEN           *
7020  C295            ;******************************
7030  C295            INDEX:
7040  C295  20FDAE    JSR CHKCOM
7050  C298  208BB0    JSR VARSUC
7060  C29B  2082C0    JSR STRPARL
7070  C29E  20FDAE    JSR CHKCOM
7080  C2A1  208BB0    JSR VARSUC
7090  C2A4  20A4C0    JSR STRPARL1
7100  C2A7  20FDAE    JSR CHKCOM
7120  C2AA  A201      LDX #1
7130  C2AC  38        SEC
7140  C2AD  A51E      LDA STRLEN1
7150  C2AF  E521      SBC STRLEN
7160  C2B1  9000    R BCC INDEX8
7170  C2B3  6900      ADC #0
7180  C2B5  8D36C0    STA ANZ
7190  C2B8            INDEX1:
7200  C2B8  A421      LDY STRLEN
7210  C2BA            INDEX2:
7220  C2BA  88        DEY
7230  C2BB  3000    R BMI INDEX9
7240  C2BD  B11F      LDA (STRADR),Y
7250  C2BF  D122      CMP (STRADR1),Y
7260  C2C1  F0F7      BEQ INDEX2
7270  C2C3  E622      INC STRADR1
7280  C2C5  D000    R BNE INDEX3
7290  C2C7  E623      INC STRADR1+1
7300  C2C9            INDEX3:
7310  C2C9  E8        INX
7320  C2CA  CE36C0    DEC ANZ
7330  C2CD  D0E9      BNE INDEX1
7340  C2CF            INDEX8:
```

```
7350  C2CF  A200     LDX #0
7360  C2D1           INDEX9:
7370  C2D1  8E36C0   STX ANZ
7375  C2D4  208BB0   JSR VARSUC
7380  C2D7  AC36C0   LDY ANZ
7390  C2DA  20A2B3   JSR YFLP
7400  C2DD  A647     LDX VARADL
7410  C2DF  A448     LDY VARADL+1
7420  C2E1  20D4BB   JSR FACXY
7430  C2E4  60       RTS
7440  C2E5           ;
```

Um Fehlern vorzubeugen, wollen wir bei diesem Unterprogramm die Parameter des Aufrufes mitangeben. Die Syntax des SYS-Aufrufs ist:

SYS 49173,A$,AA$,A

Dabei soll A$ die Zeichenreihe sein, deren Position in AA$ gesucht werden soll. Die Position wird in der Variablen A zurückgegeben; A=0 bedeutet, daß A$ nicht in AA$ enthalten ist.

Das Lesen der String-Parameter erfolgt durch den Aufruf der Routine VARSUC und anschließendem Lesen des Variableninhaltes. Durch dieses Vorgehen bedingt, können nur Variable als Parameter übergeben werden. Diese Version hat aber den Vorteil, daß die Basic-Stringverwaltung nicht benötigt wird und deshalb das Programm schneller läuft Die Adresse der ersten Zeichenreihe wird in $1F,$20 abgelegt, dessen Länge in $21, die Adresse der zweiten Zeichenreihe in $22, $23 und dessen Länge in $1E.

Dann wird die Länge der zu suchenden Zeichenreihe von der Länge der Zeichenreihe, in der gesucht werden soll, abgezogen. Ist das Ergebnis negativ, so wird zum Ende des Unterprogramms gesprungen, in dem der Ausgabeparameter auf Null gesetzt wird und in die Variable transferiert wird. Ansonsten wird die um eins erhöhte Differenz als Schleifenobergrenze ANZ gespeichert. Die Schleife, die für jede Position untersucht, ob der zu suchende String gleich dem Teilstring ab der aktuellen Stelle ist, arbeitet mit zwei Laufvariablen (ANZ und X-Register), die gegeneinander laufen. Der Abbruch erfolgt, wenn ANZ gleich 0 ist, oder der String gefunden wurde. In diesem Fall enthält das X-Register die gefundene Position.

Das Y-Register wird als Laufvariable innerhalb des zu suchenden Strings verwendet. Begonnen wird mit dem Vergleich des letzten Zeichens, und bei Übereinstimmung wird sofort

das vorhergehende Zeichen untersucht, bis die gesamte Zeichenreihe bearbeitet ist, oder eine Nicht-Übereinstimmung gefunden wurde.

Die Zuweisung des Ergebnisses an die als Parameter angegebene Variable erfolgt ab dem Label INDEX9. Der gefundene Wert wird zunächst in ANZ zwischengespeichert, die Variablenadresse mit Hilfe von VARSUC bestimmt, dann der Wert mit Hilfe von YFLFP in eine Fließkommazahl verwandelt, und schließlich diese Fließkommazahl mit Hilfe der Routine FACXY in die Variable selbst transferiert.

4.10 Sonderzeichen suchen

```
7500  C2E5                ;******************************
7510  C2E5                ;*  SONDERZEICHEN SUCHEN      *
7520  C2E5                ;******************************
7530  C2E5                SONDZ:
7540  C2E5  20FDAE        JSR CHKCOM
7550  C2E8  208BB0        JSR VARSUC
7560  C2EB  2082C0        JSR STRPARL
7570  C2EE  20FDAE        JSR CHKCOM
7580  C2F1  208BB0        JSR VARSUC
7590  C2F4  460F          LSR HOCHKM
7600  C2F6  A905          LDA #5
7610  C2F8  8D36C0        STA ANZ
7620  C2FB  20FDAE        JSR CHKCOM
7630  C2FE                SOND0:
7640  C2FE  A000          LDY #0
7650  C300  B17A          LDA (BASBZ),Y
7660  C302  F000       R  BEQ SOND9
7670  C304                SOND1:
7680  C304  20F2C0        JSR PZCODE
7690  C307  D11F          CMP (STRADR),Y
7700  C309  F000       R  BEQ SOND2
7710  C30B  C8            INY
7720  C30C  C421          CPY STRLEN
7730  C30E  90F4          BCC SOND1
7740  C310  A0FF          LDY #$FF
7750  C312                SOND2:
7760  C312  C8            INY
7770  C313  20A2B3        JSR YFLP
7780  C316  A647          LDX VARADL
7790  C318  A448          LDY VARADL+1
7800  C31A  20D4BB        JSR FACXY
7810  C31D  18            CLC
7820  C31E  A547          LDA VARADL
7830  C320  6D36C0        ADC ANZ
7840  C323  8547          STA VARADL
```

```
7850  C325  9000   R BCC SOND3
7860  C327  E648     INC VARADL+1
7870  C329           SOND3:
7880  C329  E67A     INC BASBZ
7890  C32B  D000   R BNE SOND4
7900  C32D  E67B     INC BASBZ+1
7910  C32F           SOND4:
7920  C32F  4CFEC2   JMP SOND0
7930  C332           ;
7940  C332           SOND9:
7950  C332  60       RTS
7960  C333           ;
```

Dieses Unterprogramm durchsucht eine Zeichenreihe nach einer vorgegebenen Reihe von Sonderzeichen und legt die Positionen dieser Sonderzeichen in einem Feld ab einem angegebenen Element ab. Der Aufruf sieht demnach etwa so aus:

SYS 49177,A$,A(0),"#/.,!=-;:?/&'()*

Zu Beginn des Unterprogramms werden die Daten der String-Variable eingelesen. Dann wird die Adresse des ersten Feldelementes mit Hilfe von VARSUC bestimmt. Das folgende Komma im Basic-Text wird mit CHKCOM überlesen. Dann wird das nächste Zeichen des Basic-Textes gelesen, und überprüft ob es das 0-Byte ist. Wenn ja, wird zu SOND9 gesprungen, wo das Unterprogramm beendet wird. Das angegebene Zeichen wird noch vom Interpreter-Code in den ASCII-Code mit der Routine PZCODE umgewandelt und anschließend mit jedem Zeichen des Such-Strings verglichen. Das Ergebnis wird in die angegebene Variable abgelegt und der Variablenzeiger um 5 erhöht. Die '5' ist hier in der Variablen ANZ zwischengespeichert. Dieser Wert gilt dann, wenn das Feld aus Fließkommazahlen besteht. Wenn es aus Integerzahlen besteht, muß der Wert auf '2' geändert werden.

Dann wird der Basic-Befehlszeiger um eins erhöht, womit auf das nächste zu untersuchende Sonderzeichen gezeigt und die Schleife weiter bearbeitet wird.

4.11 Fehler registrieren

```
8000  C333           ;******************************
8010  C333           ;* FEHLER REGISTRIEREN        *
8020  C333           ;******************************
8050  C333           FEHLREG:
```

```
8060  C333  209BB7    JSR GETBYT
8070  C336            FEHLER:
8080  C336  A912      LDA #18
8090  C338  AC38C0    LDY FEHLANZ
8100  C33B  C00F      CPY #15
8110  C33D  B000    R BCS FEHLREG1
8120  C33F  EE38C0    INC FEHLANZ
8130  C342  8A        TXA
8140  C343            FEHLREG1:
8150  C343  9939C0    STA FEHLERT,Y
8160  C346  60        RTS
8170  C347            ;
```

Dieses Unterprogramm wird zwar selten aufgerufen und könnte ebenso in Basic geschrieben bleiben, doch ist es sinnvoll dieses Programm in Maschine zu schreiben, damit es von Maschinenprogrammen aus aufgerufen werden kann. Die Fehlertabelle FEHLTAB ist bereits am Anfang unter Hilfszellen deklariert, ebenso die Anzahl der Fehler (FEHLANZ).

Das Unterprogramm liest aus dem Basic-Text die Nummer des Fehlers ein und speichert ihn in der Fehlertabelle, wenn nicht die maximale Anzahl der Fehler von 15 überschritten wurde. Wenn diese Zahl erreicht ist, wird anstatt des letzten Fehlers die Fehlermeldung 18 (zuviele Fehler) gespeichert.

4.12 Verwaltung der Symboltabelle

In diesem Kapitel wollen wir mehrere Unterprogramme beschreiben, welche zur Vewaltung der Symboltabelle dienen. Das ist zum einen das Umwandeln des Symbolwertes in eine Fließkommazahl, dann das Suchen eines Symbols in der Tabelle, das Übergeben der gefundenen Werte an eine Variable, das Einfügen eines neuen Symbols, das Drucken der Symboltabelle, sowie das Speichern der Tabelle auf einen externen Speicher.

Die Symboltabelle kann bis zu 255 Symbole mit ihren Werten und Typen aufnehmen. Sie besteht aus drei Bereichen: Im Bereich $9800 bis $98FF werden die Symbole jeweils mit acht Byte gespeichert. Dadurch muß ein Name immer genau acht Zeichen lang sein. Im Bereich von $9600 bis 97FF werden die Werte der Symbole jeweils als Lower- und Higher-Byte gespeichert. Schließlich werden die Typen der Symbole von %9500 bis $95FF gespeichert, dabei bedeuten:

0 - Symbol ist undefiniert
1 - Symbol ist eine Konstante
2 - Symbol ist eine Marke (Label)

Die Nummer des Symbols, aus welcher sich die Stelle berechnet, wo es gespeichert ist, wird in den folgenden Unterprogrammen immer mit SYMNR bezeichnet.

Symbolwert in Fließkommazahl umwandeln

```
10000  C347              ;*******************************
10010  C347              ;* SYMOBOLTABELLE VERWALTEN *
10020  C347              ;*******************************
10030  C347              TWERTFLP:
10040  C347   AC4CC0     LDY SYMWERT
10050  C34A   AD4DC0     LDA SYMWERT+1
10060  C34D   08         PHP
10070  C34E   2091B3     JSR AYFLP
10080  C351   28         PLP
10090  C352   1000     R BPL TWERT1
10100  C354   A94F       LDA #F65536L
10110  C356   A0C0       LDY #F65536H
10120  C358   2067B8     JSR FPLUSK
10130  C35B              TWERT1:
10140  C35B   60         RTS
```

Der Wert des Symbols wird in den Zellen SYMWERT und SYMWERT+1 übergeben. Zur Umwandlung wird die Routine AYFLP verwendet, welche aber die beiden Bytes im Akku und im Y-Register als vorzeichenbehaftete 16-Bit-Zahl auffasst. Deswegen muß noch 65536 addiert werden, wenn das MSB im Akku gesetzt war. Die Routine gibt das Ergebnis im FAC zurück.

Ausgabe eines Symbolnamens auf aktuelles Ausgabegerät

```
10160  C35C              NAMOUT:
10170  C35C   A913       LDA #NAMTAB/2048
10180  C35E   8523       STA STRADR1+1
10190  C360   AD4EC0     LDA SYMNR
10200  C363   0A         ASL A
10210  C364   2623       ROL STRADR1+1
10220  C366   0A         ASL A
10230  C367   2623       ROL STRADR1+1
10240  C369   0A         ASL A
10250  C36A   2623       ROL STRADR1+1
10260  C36C   8522       STA STRADR1
10270  C36E   A208       LDX #8
10280  C370   2025AB     JSR STROUT
10290  C373   60         RTS
```

In diesem Unterprogramm wird ein Symbol, dessen Nummer in der Zeile SYMNR übergeben wird, mit Hilfe der Routine STROUT ausgegeben. Diese Routine verlangt als Eingabeparameter die Startadresse des Strings in den Zellen $22, $23 sowie die Länge im X-Register. Da ein Symbol acht Zeichen lang ist, erhält man die Adresse des Symbols durch Addition der achtfachen Symbolnummer zur Basisadresse. Das High-Byte der Basisadresse wird durch acht geteilt und in die Zelle STRADR1+1 geschrieben. Wenn es anschließend zusammen mit der Symbolnummer um drei Bit nach links geschoben wird, also mit acht multipliziert wird, ist der Wert wieder korrekt.

CR ausgeben / Komma ausgeben

```
10310  C374            CROUT:
10320  C374  A90D      LDA #$0D ;CARRIAGE RETURN
10330  C376  200CE1    JSR BSOUT
10340  C379  60        RTS
10350  C37A            ;
10360  C37A            COMOUT:
10370  C37A  A92C      LDA #$2C ;KOMMA
10380  C37C  200CE1    JSR BSOUT
10390  C37F  60        RTS
```

Diese beiden Routinen laden den Akku mit den auszugebenden Zeichen und rufen die Routine BSOUT auf.

Typ eines Symbols ausgeben

```
10410  C380            TYPZ:
10420  C380  55434C    %ASC "UCL"
10430  C383            ;
10440  C383            TYPOUT:
10450  C383  AE4EC0    LDX SYMNR
10460  C386  BD0095    LDA TYPTAB,X
10470  C389  AA        TAX
10480  C38A  BD80C3    LDA TYPZ,X
10490  C38D  200CE1    JSR BSOUT
10500  C390  60        RTS
10510  C391            ;
```

Der Typ des Symbols wird hier als Buchstabe ausgegeben; den Typwerten 0, 1, 2 entsprechen die Zeichen U, C und L.

Wert eines Symbols dezimal ausgeben

```
10520  C391           VALOUT:
10530  C391  A94B     LDA #VALTAB/512
10540  C393  8563     STA WH
10550  C395  AD4EC0   LDA SYMNR
10560  C398  0A       ASL A
10570  C399  8562     STA WL
10580  C39B  2663     ROL WH
10590  C39D  A000     LDY #0
10600  C39F  B162     LDA (WL),Y
10610  C3A1  8D4CC0   STA SYMWERT
10620  C3A4  C8       INY
10630  C3A5  B162     LDA (WL),Y
10640  C3A7  8D4DC0   STA SYMWERT+1
10650  C3AA  2047C3   JSR TWERTFLP
10660  C3AD  20DDBD   JSR $BDDD ; FAC IN ASCII WANDELN
10670  C3B0  201EAB   JSR $AB1E ; STRING AUSGEBEN
10680  C3B3  60       RTS
10690  C3B4           ;
```

Der Wert des Symbols wird ähnlich wie der Name des Symbols geholt und in den Zellen SYMWERT und SYMWERT+1 zwischengespeichert. Dann werden Akku und Y-Register mit den Inhalten dieser Zellen geladen und die Routine AYFLP aufgerufen, welche die Zahl im Fließkomma-Akkumulator ablegt. Mit Hilfe der darauf folgenden beiden Unterprogammaufrufe wird schließlich die Zahl als Zeichenreihe ausgegeben.

Wert eines Symbols hexadezimal ausgeben

```
10700  C3B4           VALHOUT:
10710  C3B4  A94B     LDA #VALTAB/512
10720  C3B6  8563     STA WH
10730  C3B8  AD4EC0   LDA SYMNR
10740  C3BB  0A       ASL A
10750  C3BC  8562     STA WL
10760  C3BE  2663     ROL WH
10770  C3C0  A001     LDY #1
10780  C3C2  B162     LDA (WL),Y
10790  C3C4  8D37C0   STA WERT
10800  C3C7  2025C2   JSR HEX
10810  C3CA  AD0002   LDA BEP
10820  C3CD  200CE1   JSR BSOUT
10830  C3D0  AD0102   LDA BEP+1
10840  C3D3  200CE1   JSR BSOUT
10850  C3D6  A000     LDY #0
```

```
10860  C3D8  B162      LDA (WL),Y
10870  C3DA  8D37C0    STA WERT
10880  C3DD  2025C2    JSR HEX
10890  C3E0  AD0002    LDA BEP
10900  C3E3  200CE1    JSR BSOUT
10910  C3E6  AD0102    LDA BEP+1
10920  C3E9  200CE1    JSR BSOUT
10930  C3EC  60        RTS
10940  C3ED            ;
```

Diese Ausgabe ist ähnlich der dezimalen Ausgabe, jedoch wird zur Umwandlung jeweils die Routine HEX aufgerufen, und die erhaltenen Bytes werden sofort ausgegeben.

Symbol in Tabelle suchen

```
10950  C3ED              SYMSUCH:
10960  C3ED  A000        LDY #0
10970  C3EF  8C4EC0      STY SYMNR
10980  C3F2  20FDAE      JSR CHKCOM
10990  C3F5  209EAD      JSR FRMEVL
11000  C3F8  20A3B6      JSR FRESTR
11010  C3FB  C908        CMP #8
11020  C3FD  D000     R  BNE SYMERR
11030  C3FF  AD49C0      LDA SYMANZ
11040  C402  F000     R  BEQ SYMNOTF
11050  C404              SYM1:
11060  C404  A913        LDA #NAMTAB/2048
11070  C406  8563        STA WH
11080  C408  AD4EC0      LDA SYMNR
11090  C40B  0A          ASL A
11100  C40C  2663        ROL WH
11110  C40E  0A          ASL A
11120  C40F  2663        ROL WH
11130  C411  0A          ASL A
11140  C412  2663        ROL WH
11150  C414  8562        STA WL
11160  C416  A000        LDY #0
11170  C418              SYM2:
11180  C418  B162        LDA (WL),Y
11190  C41A  D122        CMP (STRADR1),Y
11200  C41C  D000     R  BNE SYM3
11210  C41E  C8          INY
11220  C41F  C008        CPY #8
11230  C421  D0F5        BNE SYM2
11240  C423              SYMFOUND:
11250  C423  18          CLC
11260  C424  60          RTS
11270  C425
```

```
11280  C425              SYM3:
11290  C425   AC4EC0     LDY SYMNR
11300  C428   C8         INY
11310  C429   CC49C0     CPY SYMANZ
11320  C42C   8C4EC0     STY SYMNR
11330  C42F   D0D3       BNE SYM1
11340  C431              SYMNOTF:
11350  C431   38         SEC
11360  C432   60         RTS
11370  C433              ;
11380  C433              SYMERR:
11390  C433   20FDAE     JSR CHKCOM
11400  C436   20EBB7     JSR GETAB
11410  C439   4C48B2     JMP ILLQERR
11420  C43C              ;
```

Zunächst wird das zu suchende Symbol aus dem Basic-Text mit Hilfe von FRMEVL und FRESTR gelesen. Die Länge des Strings wird dabei im Akku übergeben. Ist sie ungleich acht, so wird zu einer Fehlermeldung verzweigt.

Wenn die Befehlstabelle leer ist (SYMANZ=0), so wird zum Ende des Programms gesprungen, wobei noch das Carry-Flag gesetzt wird, um anzuzeigen, daß das Symbol nicht gefunden wurde.

Im anderen Fall wird die Startadresse des Symbols berechnet (s.o.), dann wird in einer Schleife mit dem Y-Register ein Zeichen nach dem anderen verglichen. Wurde die Schleife beendet, ohne daß eine Nichtübereinstimmung festgestellt wurde, so wurde das Symbol gefunden und das Unterprogramm mit gelöschtem Carry-Flag verlassen.

Im anderen Fall wird der Zeiger WL,WH auf das nächste Symbol gerichtet, solange die Symboltabelle noch nicht ausgeschöpft ist. Wenn SYMNR gleich SYMANZ ist, wurden alle Symbole überprüft; daraus folgt, daß das Symbol nicht in der Tabelle enthalten ist.

Symbol suchen und Werte an Variable zuweisen

```
11430  C43C              TABSUCH:
11440  C43C   20EDC3     JSR SYMSUCH
11450  C43F   B000     R BCS TABSNG
11460  C441   AE4EC0     LDX SYMNR
11470  C444   8662       STX WL
```

```
11480  C446  A04B      LDY #VALTAB/512
11490  C448  8463      STY WH
11500  C44A  0662      ASL WL
11510  C44C  2663      ROL WH
11520  C44E  A000      LDY #0
11530  C450  B162      LDA (WL),Y
11540  C452  8D4CC0    STA SYMWERT
11550  C455  C8        INY
11560  C456  B162      LDA (WL),Y
11570  C458  8D4DC0    STA SYMWERT+1
11580  C45B  BD0095    LDA TYPTAB,X
11590  C45E  8D4BC0    STA SYMTYP
11600  C461            ;
11610  C461            TABWERT:
11620  C461  20FDAE    JSR CHKCOM
11630  C464  208BB0    JSR VARSUC
11640  C467  2047C3    JSR TWERTFLP
11650  C46A  A647      LDX VARADL
11660  C46C  A448      LDY VARADL+1
11670  C46E  20D4BB    JSR FACXY
11680  C471  20FDAE    JSR CHKCOM
11690  C474  208BB0    JSR VARSUC
11700  C477  AC4BC0    LDY SYMTYP
11710  C47A  20A2B3    JSR YFLP
11720  C47D  A647      LDX VARADL
11730  C47F  A448      LDY VARADL+1
11740  C481  20D4BB    JSR FACXY
11750  C484  60        RTS
11760  C485            ;
11770  C485            TABSNG:
11780  C485  A900      LDA #0
11790  C487  8D4CC0    STA SYMWERT
11800  C48A  8D4DC0    STA SYMWERT+1
11810  C48D  8D4BC0    STA SYMTYP
11820  C490  F0CF      BEQ TABWERT
11830  C492            ;
```

Zunächst wird das Symbol mit Hilfe von SYMSUCH gesucht und anschließend der Wert und der Typ des Symbols in die Zellen SYMWERT, SYMWERT+1 und SYMTYP abgelegt, wenn das Symbol gefunden wurde. Ist es nicht gefunden, so werden diese Zellen mit Null belegt. Die Werte aus diesen Zellen werden anschließend in die Variablen übertragen.

Symbol in Tabelle einfügen

```
11840  C492            TABEINF:
11850  C492  20EDC3    JSR SYMSUCH
11860  C495  9000    R BCC TABEF2
```

```
11870  C497  AD49C0     LDA SYMANZ
11880  C49A  EE49C0     INC SYMANZ
11890  C49D  F000     R BEQ TABEF1
11900  C49F             TABE1:
11910  C49F  A213       LDX #NAMTAB/2048
11920  C4A1  8663       STX WH
11930  C4A3  0A         ASL A
11940  C4A4  2663       ROL WH
11950  C4A6  0A         ASL A
11960  C4A7  2663       ROL WH
11970  C4A9  0A         ASL A
11980  C4AA  2663       ROL WH
11990  C4AC  8562       STA WL
12000  C4AE  A007       LDY #$07
12010  C4B0             TABE2:
12020  C4B0  B122       LDA (STRADR1),Y
12030  C4B2  9162       STA (WL),Y
12040  C4B4  88         DEY
12050  C4B5  10F9       BPL TABE2
12060  C4B7  20FDAE     JSR CHKCOM
12070  C4BA  20EBB7     JSR GETAB
12080  C4BD  8A         TXA
12090  C4BE  AE49C0     LDX SYMANZ
12100  C4C1  CA         DEX
12110  C4C2  9D0095     STA TYPTAB,X
12120  C4C5  8662       STX WL
12130  C4C7  A24B       LDX #VALTAB/512
12140  C4C9  8663       STX WH
12150  C4CB  0662       ASL WL
12160  C4CD  2663       ROL WH
12170  C4CF  A000       LDY #0
12180  C4D1  A514       LDA ADRL
12190  C4D3  9162       STA (WL),Y
12200  C4D5  C8         INY
12210  C4D6  A515       LDA ADRH
12220  C4D8  9162       STA (WL),Y
12230  C4DA  60         RTS
12240  C4DB             ;
12250  C4DB             TABEF1:
12260  C4DB  A216       LDX #22 ;SYMBOLTABELLE VOLL
12270  C4DD  2036C3     JSR FEHLER
12280  C4E0  A9FF       LDA #$FF
12290  C4E2  8D49C0     STA SYMANZ
12300  C4E5  60         RTS
12310  C4E6             ;
12320  C4E6             TABEF2:
12330  C4E6  A20B       LDX #11
12340  C4E8  2036C3     JSR FEHLER
12350  C4EB  20FDAE     JSR CHKCOM
12360  C4EE  20EBB7     JSR GETAB
12370  C4F1  60         RTS
12380  C4F2
```

Dieses Unterprogramm beinhaltet den umgekehrten Vorgang wie das Unterprogramm TABSUCH, dabei wird jedoch eine Fehlermeldung ausgegeben, wenn das Symbol bereits in der Tabelle enthalten ist.

Tabelle drucken

```
12390  C4F2             TABDRUCK:
12400  C4F2  AD49C0     LDA SYMANZ
12410  C4F5  F000     R BEQ TABDEND
12420  C4F7  A200       LDX #0
12430  C4F9             TABD1:
12440  C4F9  8E4EC0     STX SYMNR
12450  C4FC  205CC3     JSR NAMOUT
12460  C4FF  203FAB     JSR SPOUT
12470  C502  2083C3     JSR TYPOUT
12480  C505  203FAB     JSR SPOUT
12490  C508  20B4C3     JSR VALHOUT
12500  C50B  2074C3     JSR CROUT
12510  C50E  AE4EC0     LDX SYMNR
12520  C511  E8         INX
12530  C512  EC49C0     CPX SYMANZ
12540  C515  90E2       BCC TABD1
12550  C517             TABDEND:
12560  C517  60         RTS
12570  C518             ;
```

In einer Schleife mit der Laufvariablen SYMNR werden alle Symbole sowie deren Werte und Typen nacheinander - durch Leerzeichen getrennt - auf dem aktuellen Ausgabegerät ausgegeben. Die Ausgabe des Wertes erfolgt mit der Routine VALHOUT, also hexadezimal.

Tabelle speichern

```
12580  C518             TABSPEI:
12590  C518  209BB7     JSR GETBYT
12600  C51B  8613       STX AKTIO
12610  C51D  2018E1     JSR CHKOUT
12620  C520  AD49C0     LDA SYMANZ
12630  C523  F000     R BEQ TABSPEND
12640  C525  A200       LDX #0
12650  C527             TABSP1:
12660  C527  8E4EC0     STX SYMNR
12670  C52A  2083C3     JSR TYPOUT
12680  C52D  207AC3     JSR COMOUT
```

```
12690  C530  205CC3   JSR NAMOUT
12700  C533  207AC3   JSR COMOUT
12710  C536  2091C3   JSR VALOUT
12720  C539  2074C3   JSR CROUT
12730  C53C  AE4EC0   LDX SYMNR
12740  C53F  E8       INX
12750  C540  EC49C0   CPX SYMANZ
12760  C543  90E2     BCC TABSP1
12770  C545           TABSPEND:
12780  C545  A613     LDX AKTIO
12790  C547  20CCFF   JSR CLRCH
12800  C54A  A200     LDX #0
12810  C54C  8613     STX AKTIO
12820  C54E  60       RTS
12830  C54F           ;
```

Zuerst wird aus dem Basic-Text die logische Dateinummer der Datei bestimmt, auf die ausgegeben werden soll. Dann wird der Eingabekanal für diese Datei geöffnet.

Im übrigen gleicht diese Routine im wesentlichen der oben beschriebenen Routine 'Tabelle drucken', jedoch werden hier die Werte durch Kommata getrennt; außerdem werden die Symbolwerte dezimal gespeichert. Zum Schluß wird der Eingabekanal wieder geschlossen.

4.13 Mnemotechnische Bezeichnung suchen

```
15000  C54F           ;*****************************
15010  C54F           ;* TABELLE DER MNEMONICS     *
15020  C54F           ;*****************************
15030  C54F           MNEMOTAB:
15100  C54F  414443   %ASC "ADC"
15101  C552  414E44   %ASC "AND"
15102  C555  41534C   %ASC "ASL"
15103  C558  424343   %ASC "BCC"
15104  C55B  424353   %ASC "BCS"
15105  C55E  424551   %ASC "BEQ"
15106  C561  424954   %ASC "BIT"
15107  C564  424D49   %ASC "BMI"
15108  C567  424E45   %ASC "BNE"
15109  C56A  42504C   %ASC "BPL"
15110  C56D  42524B   %ASC "BRK"
15111  C570  425643   %ASC "BVC"
15112  C573  425653   %ASC "BVS"
15113  C576  425954   %ASC "BYT"
15114  C579  434C43   %ASC "CLC"
15115  C57C  434C44   %ASC "CLD"
```

```
15116  C57F  434C49    %ASC "CLI"
15117  C582  434C56    %ASC "CLV"
15118  C585  434D50    %ASC "CMP"
15119  C588  435058    %ASC "CPX"
15120  C58B  435059    %ASC "CPY"
15121  C58E  444543    %ASC "DEC"
15122  C591  444558    %ASC "DEX"
15123  C594  444559    %ASC "DEY"
15124  C597  454F52    %ASC "EOR"
15125  C59A  494E43    %ASC "INC"
15126  C59D  494E58    %ASC "INX"
15127  C5A0  494E59    %ASC "INY"
15128  C5A3  4A4D50    %ASC "JMP"
15129  C5A6  4A5352    %ASC "JSR"
15130  C5A9  4C4441    %ASC "LDA"
15131  C5AC  4C4458    %ASC "LDX"
15132  C5AF  4C4459    %ASC "LDY"
15133  C5B2  4C5352    %ASC "LSR"
15134  C5B5  4E4F50    %ASC "NOP"
15135  C5B8  4F5241    %ASC "ORA"
15136  C5BB  504841    %ASC "PHA"
15137  C5BE  504850    %ASC "PHP"
15138  C5C1  504C41    %ASC "PLA"
15139  C5C4  504C50    %ASC "PLP"
15140  C5C7  524F4C    %ASC "ROL"
15141  C5CA  524F52    %ASC "ROR"
15142  C5CD  525449    %ASC "RTI"
15143  C5D0  525453    %ASC "RTS"
15144  C5D3  534243    %ASC "SBC"
15145  C5D6  534543    %ASC "SEC"
15146  C5D9  534544    %ASC "SED"
15147  C5DC  534549    %ASC "SEI"
15148  C5DF  535441    %ASC "STA"
15149  C5E2  535458    %ASC "STX"
15150  C5E5  535459    %ASC "STY"
15151  C5E8  544158    %ASC "TAX"
15152  C5EB  544159    %ASC "TAY"
15153  C5EE  545358    %ASC "TSX"
15154  C5F1  545841    %ASC "TXA"
15155  C5F4  545853    %ASC "TXS"
15156  C5F7  545941    %ASC "TYA"
15157  C5FA  574F52    %ASC "WOR"
15200  C5FD  00        UGR: BYT 0
15210  C5FE  39        OGR: BYT 57
15220  C5FF            MNEMO:
15230  C5FF  20FDAE    JSR CHKCOM
15240  C602  209EAD    JSR FRMEVL
15250  C605  20A3B6    JSR FRESTR
15260  C608  A000      LDY #0
15270  C60A  C903      CMP #3
15280  C60C  D000    R BNE MNEMOE
```

```
15290  C60E  8CFDC5     STY UGR
15300  C611  A939       LDA #57
15310  C613  8DFEC5     STA OGR
15320  C616             MNEMO1:
15330  C616  A000       LDY #0
15340  C618  ADFEC5     LDA OGR
15350  C61B  CDFDC5     CMP UGR
15360  C61E  9000     R BCC MNEMOE
15370  C620  18         CLC
15380  C621  6DFDC5     ADC UGR
15390  C624  4A         LSR A
15400  C625  8D37C0     STA WERT
15410  C628  0A         ASL A
15420  C629  6D37C0     ADC WERT
15430  C62C  AA         TAX
15440  C62D  A000       LDY #0
15450  C62F             MNEMO2:
15460  C62F  BD4FC5     LDA MNEMOTAB,X
15470  C632  D122       CMP (STRADR1),Y
15480  C634  D000     R BNE MNEMO3
15490  C636  E8         INX
15500  C637  C8         INY
15510  C638  C003       CPY #3
15520  C63A  D0F3       BNE MNEMO2
15530  C63C  AC37C0     LDY WERT
15540  C63F  C8         INY
15550  C640             MNEMOE:
15560  C640             ;ERGEBNIS STEHT IM Y-REGISTER
15570  C640  60         RTS
15580  C641             ;
15590  C641             MNEMO3:
15600  C641  B000     R BCS MNEMO4
15610  C643  AC37C0     LDY WERT
15620  C646  C8         INY
15630  C647  8CFDC5     STY UGR
15640  C64A  D0CA       BNE MNEMO1
15650  C64C             MNEMO4:
15660  C64C  AC37C0     LDY WERT
15670  C64F  F0EF       BEQ MNEMOE
15680  C651  88         DEY
15690  C652  8CFEC5     STY OGR
15700  C655  4C16C6     JMP MNEMO1
15710  C658             ;
```

Zu Beginn dieses Programmstücks steht die Tabelle der mnemotechnischen Bezeichnungen ab dem Label MNEMOTAB. Die Tabelle konnte einheitlich aufgebaut werden, da beim 6510 jedes Mnemonic genau drei Zeichen lang ist. Anschließend werden zwei Hilfszellen UGR und OGR definiert, die zum binären Suchen in der Tabelle gebraucht werden. Der Aufruf

des Unterprogramms zielt auf das Label MNEMO. Hier ist ein Beispiel zu sehen, wie ein String-Parameter mit den Routinen FRMEVL und FRESTR eingelesen wird. Nach dem Aufruf dieser beiden Routinen steht die String-Adresse in $22 und $23, die Länge des Strings wird im Akku übergeben. Wenn die Länge nicht 3 ist, wird zum Label MNEMOE verzweigt, welches das Unterprogramm beendet. Grundsätzlich wird die gefundene Position im Y-Register übergeben, wodurch sie im Basic-Programm durch den Befehl PEEK(782) abgefragt werden kann.

Zum Suchen wird, wie oben erwähnt, der Algorithmus des binären Suchens verwendet. D.h. der zu suchende Wert wird zunächst mit dem Wert in der Mitte der Tabelle verglichen. Ist der Wert dann gefunden, ist alles okay. Wenn der zu suchende Wert kleiner ist als der mittlere Wert, wird in der ersten Hälfte der Tabelle weiter gesucht, sonst in der zweiten Hälfte. Dabei wird jeweils wieder die Mitte der Hälfte angesteuert. Durch fortgesetztes Halbieren verkleinern sich die durchzusuchenden Teilbereiche schnell.

Die aktuell zu durchsuchende Position wird in der Hilfszelle WERT zwischengespeichert. Wurde der Wert nicht gefunden, so wird entweder die Untergrenze auf den Wert (WERT)+1 oder die Obergrenze auf den Wert (WERT)-1 gesetzt. Das Ende einer nicht erfolgreichen Suche kann daran erkannt werden, daß die Untergrenze größer als die Obergrenze ist.

4.14 Initialisierungs-Routine

```
20000  C658          ;*****************************
20010  C658          ;* INITIALISIERUNGSROUTINE   *
20020  C658          ;*****************************
20030  C658          INIT:
20040  C658          USRVH=HEXDEZ/256
20050  C658          USRV=USRVH*256
20060  C658  A98A    LDA #HEXDEZ-USRV
20070  C65A  8D1103  STA USRVEC
20080  C65D  A9C1    LDA #USRVH
20090  C65F  8D1203  STA USRVEC+1
20100  C662  A995    LDA #160-11 ;NEUES ENDRAM HIGH
20110  C664  8538    STA $37+1 ;ENDRAM HIGH
20120  C666  8534    STA $33+1 ;STARTSTR HIGH
20130  C668  A200    LDX #0
20140  C66A  8A      TXA
20150  C66B          INIT1:
20160  C66B  9D0095  STA TYPTAB,X
20170  C66E  9D0096  STA VALTAB,X
```

```
20180  C671  9D0097   STA VALTAB+256,X
20190  C674  9D0098   STA NAMTAB,X
20200  C677  9D0099   STA NAMTAB+256,X
20210  C67A  9D009A   STA NAMTAB+512,X
20220  C67D  9D009B   STA NAMTAB+768,X
20230  C680  9D009C   STA NAMTAB+1024,X
20240  C683  9D009D   STA NAMTAB+1280,X
20250  C686  9D009E   STA NAMTAB+1536,X
20260  C689  9D009F   STA NAMTAB+1792,X
20270  C68C  E8       INX
20280  C68D  D0DC     BNE INIT1
20290  C68F  8D49C0   STA SYMANZ
20300  C692  8D38C0   STA FEHLANZ
20310  C695  60       RTS
```

In diesem Programmstück sind alle Vorgänge zusammengefaßt, die zu Beginn des Programms ausgeführt werden müssen. Das ist zum einen das Setzen des USR-Vektors, dann das Beschränken des Basic-Speichers. Für die Symboltabelle müssen 11 mal 256 Byte zur Verfügung gestellt werden. Die Symboltabelle wurde in den Basic-Speicherbereich gelegt, weil im Speicher zwischen $C000 und $CFF nicht mehr genügend Platz ist, da hier auch die verwaltenden Programme liegen.

Schließlich wird die gesamte Tabelle mit Nullen belegt. Außerdem wird die Anzahl der Symbole und die Anzahl der Fehler auf Null gesetzt.

5

Disassembler

5. Disassembler

Der in diesem Buch vorgestellte Disassembler ist eine erweiterte Version des in Band 3 beschriebenen. Das vorliegende Programm erlaubt das Disassemblieren von auf Floppy gespeicherten Objekt-Dateien und Bereichen des Betriebssystems. Die Ausgabe kann wahlweise auf Bildschirm, Drucker oder auf Floppy als Assembler-Quellprogramm in Form einer sequentiellen Datei oder als Programmdatei erfolgen.

Neu ist die Verwaltung von Symbolen und die Ausgabe des disassemblierten Codes als eine Programmdatei. Dadurch ist es zum Beispiel möglich, mit dem vorgestellten Assembler ein disassembliertes Listing wieder zu assemblieren. Dies ist unter Umständen dann sinnvoll, wenn anderweitig bezogene Objektdateien ohne Source-Listing verändert werden sollen. Z.B. läßt sich so das in Band 5 vorgestellte disassemblierte Listing von Simon's Basic den eigenen Wünschen anpassen.

Die Beschreibung des vorliegenden Disassemblers geschieht in vier Abschnitten. Zunächst werden die im Programm verwendeten DATA-Anweisungen beschrieben, sodann die verwendeten Unterprogramme und anschließend das Hauptprogramm. Schließlich wird noch auf Erweiterungsmöglichkeiten eingegangen.

5.1 DATA-Anweisungen

```
50000 REM *** DATA *** MNEMONICS GEORDNET NACH CODES ***
50010 DATABRK,ORAIX,,,,ORAZ,ASLZ,,PHP,ORA#
50020 DATAASL A,,,ORA$,ASL$,,BPL+,ORAIY,,
50030 DATA,ORAZX,ASLZX,,CLC,ORA$Y,,,,ORA$X
50040 DATAASL$X,,JSR$,ANDIX,,,BITZ,ANDZ,ROLZ,
50050 DATAPLP,AND#,ROL A,,BIT$,AND$,ROL$,,BMI+,ANDIY
50060 DATA,,,ANDZX,ROLZX,,SEC,AND$Y,,
50070 DATA,AND$X,ROL$X,,RTI,EORIX,,,,EORZ
50080 DATALSRZ,,PHA,EOR#,LSR A,,JMP$,EOR$,LSR$,
50090 DATABVC+,EORIY,,,,EORZX,LSRZX,,CLI,EOR$Y
50100 DATA,,,EOR$X,LSR$X,,RTS,ADCIX,,
50110 DATA,ADCZ,RORZ,,PLA,ADC#,ROR A,,JMP$I,ADC$
50120 DATAROR$,,BVS+,ADCIY,,,,ADCZX,RORZX,
50130 DATASEI,ADC$Y,,,,ADC$X,ROR$X,,,STAIX
50140 DATA,,STYZ,STAZ,STXZ,,DEY,,TXA,
```

```
50150 DATASTY$,STA$,STX$,,BCC+,STAIY,,,STYZX,STAZX
50160 DATASTXZY,,TYA,STA$Y,TXS,,,STA$X,,
50170 DATALDY#,LDAIX,LDX#,,LDYZ,LDAZ,LDXZ,,TAY,LDA#
50180 DATATAX,,LDY$,LDA$,LDX$,,BCS+,LDAIY,,
50190 DATALDYZX,LDAZX,LDXZY,,CLV,LDA$Y,TSX,,LDY$X,LDA$X
50200 DATALDX$Y,,CPY#,CMPIX,,,CPYZ,CMPZ,DECZ,
50210 DATAINY,CMP#,DEX,,CPY$,CMP$,DEC$,,BNE+,CMPIY
50220 DATA,,,CMPZX,DECZX,,CLD,CMP$Y,,
50230 DATA,CMP$X,DEC$X,,CPX#,SBCIX,,,CPXZ,SBCZ
50240 DATAINCZ,,INX,SBC#,NOP,,CPX$,SBC$,INC$,
50250 DATABEQ+,SBCIY,,,,SBCZX,INCZX,,SED,SBC$Y
50260 DATA,,,SBC$X,INC$X,
```

Unabdingbar für die Funktion des Disassemblers ist eine Tabelle, aus der hervorgeht, in welche nmemotechnische Bezeichnung ein bestimmter Maschinen-Code zu übersetzen ist. Dabei muß außerdem festgehalten werden, ob nach dem eigentlichen Code noch ein oder zwei Byte von einem eventuellen Operanden folgen oder nicht. Für die Angabe der Adressierungsart, wurden folgende Kürzel verwendet, die an die Bezeichnung angehängt sind:

Kürzel	Adressierungsart	Operand
'IX'	indiziert indirekt	(Adresse,X)
'IY'	indirekt indiziert	(Adresse),Y
'$'	absolut	Adresse
'$X'	absolut X-indiziert	Adresse,X
'$Y'	absolut Y-indiziert	Adresse,Y
'Z'	Zeropage	Adresse
'ZX'	Zeropage X-indiziert	Adresse,X
'ZY'	Zeropage Y-indiziert	Adresse,Y
'I'	indirekt	(Adresse)
'#'	immediate	#Wert
'+'	relativ	Marke
' A'	Akkumulator	A

Ist kein solches Kürzel vorhanden, so ist die Adressierungsart implizit, d.h. zu diesem Befehl gibt es keinen Operanden.

In den DATA-Zeilen ab 50000 sind die Befehle nach Ihrem Code-Wert sortiert eingetragen. Eine leere Zeichenreihe bedeutet, daß es diesen Befehl nicht gibt. Das kann in einem Objekt-Programm vorkommen, wenn ein Text, eine Konstante oder ähnliches auftaucht.

5.2 Unterprogramme

Nächstes Byte holen

```
10 REM *** UPRO *** NEACHSTES BYTE HOLEN ***
20 A=A+1
30 IFED$=""THEN90
40 GET#2,I$
50 IS=ST
60 IFI$=""THENI$=CHR$(0)
70 II=ASC(I$)
80 RETURN
90 II=PEEK(A)
100 IS=0
110 IFA>=EATHENIS=64
120 RETURN
```

Dieses Unterprogramm stellt in der Variablen II das nächste zu disassemblierende Byte, und in der Variablen IS den Eingabestatus zur Verfügung. Außerdem wird die aktuelle Adresse A um '1' erhöht. Je nachdem, ob das Zeichen aus der Eingabedatei ED$, oder aus dem Speicher (dann ist ED$="") gelesen werden soll, wird zu den Zeilen 40 bzw. 90 verzweigt. Der Eingabestatus IS ist 0, wenn das letzte zu disassemblierende Byte noch nicht erreicht wurde, andernfalls 64.

In dem Fall, daß aus der Eingabedatei gelesen werden soll, wird IS gleich ST gesetzt, im anderen Fall wird geprüft, ob die Endadresse EA des zu übersetzenden Speicherabschnittes erreicht ist.

Zweistellige Hexzahl bilden

```
200 REM *** UPRO *** 2-STELLIGE HEXZAHL (H$) AUS H BILDEN
210 H$=MID$(HE$,H/16+1,1)+MID$(HE$,(HAND15)+1,1)
220 RETURN
```

Die Variable HE$, die in Zeile 1050 besetzt wird, enthält die Zeichen "0123456789ABCDEF". Jeweils für die höherwertigen vier Bit (H/16) und die niederwertigen vier Bit (H AND 15) wird ein Zeichen aus HE$ ausgewählt.

Vierstellige Hexzahl bilden

```
250 REM *** UPRO *** 4-STELLIGE HEXZAHL (HH$) AUS HH BILDEN
260 H=INT(HH/256)
270 GOSUB200
280 HH$=H$
290 H=HH-256*H
300 GOSUB200
310 HH$=HH$+H$
320 RETURN
```

Hier wird die Variable HH in ein höherwertiges Byte und ein niederwertiges Byte zerlegt, für welche jeweils das oben beschriebene Unterprogramm aufgerufen wird. Das Ergebnis wird in der Variablen HH$ zusammengefaßt.

Ein-Byte-Operand bilden

```
500 REM *** 1-BYTE-OPERAND BILDEN ***
510 IFPR$<>"$"THEN560
520 W=H:GOSUB800
530 IFTN$=""ORTY$<>"C"THEN560
540 H$=TN$
550 RETURN
560 IFHD$="D"THENH$=MID$(STR$(H),2):RETURN
570 GOSUB200
580 H$=PR$+H$
590 RETURN
```

In der Variablen HD$ wird übergeben, ob der Operand dezimal (HD$="D") dargestellt werden soll, oder hexadezimal (HD$="H"). Die Variable PR$ gibt an, ob der Operand für die Darstellung eines Befehlscodes (PR$=" ") oder einen Operanden (PR$="$") gebraucht wird. Im letzeren Fall wird anstatt einer Zahl ein Symbolname zurückgegeben, wenn die umzuwandelnde Adresse in der eingelesenen Tabelle enthalten ist.

Zwei-Byte-Operand bilden

```
600 REM *** 2-BYTE-OPERAND BILDEN ***
610 IFPR$<>"$"THEN660
620 W=HH:GOSUB800
630 IFTN$=""THEN660
640 HH$=TN$
650 RETURN
660 IFHD$="D"THENHH$=MID$(STR$(HH)+"      ",2,5):RETURN
670 GOSUB250
680 HH$=PR$+HH$
690 RETURN
```

Dieses Unterprogramm funktioniert analog zum oben beschriebenen.

Wert in Tabelle suchen

```
800 REM *** WERT SUCHEN IN TABELLE ***
810 TN$=""
820 IFTA=0THENRETURN
830 FORI=1TOTA
840 IFTV(I)<>WTHENNEXT
850 IFI<=TATHENTN$=TN$(I):TY$=TY$(I)
860 RETURN
```

Wenn der Symbolwert W in der Tabelle der Symbole enthalten ist, wird den Variablen TN$ und TY$ der Name und der Typ des gefundenen Symbols zugeordnet. Ist kein Symbol mit dem gesuchten Wert vorhanden, so wird TN$ auf "" gesetzt.

Disk-Status-Werte bestimmen

```
900 REM *** UPRO *** DISK-STATUS-WERTE DS UND DS$ BESTIMMEN
910 INPUT#15,DS,DS$,D1$,D2$
920 DS$=STR$(DS)+","+DS$+","+D1$+","+D2$
930 RETURN
```

Der Status wird zunächst aus dem Fehlerkanal der Floppy in die Variablen DS,DS$,D1$ und D2$ eingelesen. Anschließend wird noch die Variable DS$ aus diesen zusammengesetzt.

Eine Zeile in Datei schreiben

```
3000 REM *** UPRO EINE ZEILE D$ IN DATEI SCHREIBEN
3010 ZN=ZN+10
3020 IFPA=0THENPRINT#6,D$:RETURN
3030 PA=PA+LEN(D$)+5
3040 HI=INT(PA/256):LO=PA-256*HI
3050 PRINT#6,CHR$(LO)CHR$(HI);
3060 HI=INT(ZN/256):LO=ZN-256*HI
3070 PRINT#6,CHR$(LO)CHR$(HI);D$;CHR$(0);
3080 RETURN
```

In diesem kurzen Unterprogramm wird die laufende Zeilennummer für die Ausgabedatei um 10 erhöht; das fertige Listing ist somit in 10-er Schritten durchnumeriert. Dann wird gefragt, ob auf eine Programmdatei (PA größer 0) oder eine sequentielle Datei (PA=0) gelistet werden soll. Im ersten Fall muß noch der Vorwärtszeiger auf die nächste Programmzeile berechnet werden, außerdem muß die Variable PA um die Länge von D$ + 5 Byte (je 2 für Vorwärtszeiger und Zeilennummer und 1 Byte für das Abschlußbyte CHR$(0)) erhöht werden.

Liegt eine sequentielle Ausgabedatei vor, so braucht nur der Text selbst gespeichert zu werden.

5.3 Hauptprogramm

```
1000 DIMCD$(255),TN$(500),TY$(500),TV(500)
1010 FORI=0TO255
1020 READCD$(I)
1030 NEXT
1040 OPEN15,8,15
1050 HE$="0123456789ABCDEF"
1060 SD$=""
1070 INPUT"SYMBOLTABELLE VON DATEI     ■■■I";SD$
1080 IFSD$=" "THEN1190
1090 OPEN5,8,5,SD$
1100 GOSUB900
1110 IFDSTHEN1170
1120 TA=TA+1
1130 INPUT#5,TY$(TA),TN$(TA),TV(TA)
1135 TY$(TA)=LEFT$(TY$(TA),1)
1140 IS=ST
1150 IFST=2THENTA=TA-1
1160 IFST=0THEN1120
1170 CLOSE5
1180 GOTO1060
1190 ED$=""
1200 INPUT"EINGABEDATEI  ";ED$
1210 IFED$>""THEN1260
1220 INPUT"STARTADRESSE, ENDADRESSE";SA,EA
1230 IFEA=0THEN2590
1240 IFATHEN1600
1250 GOTO1290
1260 OPEN2,8,2,ED$
1270 GOSUB900
1280 IFDSTHENPRINTDS$:CLOSE2:GOTO1190
1290 INPUT"AUSGABEGERAET ";AG
1300 IFAG<8THENOPEN3,AG:GOTO1430
1310 OPEN3,
```

```
1320 INPUT"AUSGABEDATEI, TYP (P,S,U) ";AD$,AT$
1330 OPEN6,AG,6,AD$+","+AT$+",W"
1340 GOSUB900
1350 IFDS<>63THEN1410
1360 INPUT"ALTE VERSION LOESCHEN     N███";A$
1370 IFA$<>"J"GOTO1410
1380 CLOSE6
1390 PRINT#15,"S:"+AD$
1400 GOTO1330
1410 IFDSTHENPRINTDS$:CLOSE6:GOTO1320
1420 IFAT$="P"THENPA=2049:PRINT#6,CHR$(1)CHR$(8);
1430 PRINT"HEXADEZIMAL/DEZIMAL (H/D) ? ";
1440 GETHD$
1450 IFHD$<>"H"ANDHD$<>"D"THEN1440
1460 PRINTHD$
1470 IFAG<8ORTA=0THEN1550
1480 FORI=1TOTA
1490 IFTY$(I)="L"THEN1540
1500 HH=TV(I):GOSUB600
1510 IFHD$="H"THENHH$="$"+HH$
1520 D$=TN$(I)+"="+HH$
1530 GOSUB3000
1540 NEXT
1550 IFED$=""THEN1600
1560 GOSUB10
1570 SA=II
1580 GOSUB10
1590 SA=SA+256*II
1600 A=SA
1610 IFAG<8GOTO2000
1620 HH=SA
1630 GOSUB600
1640 IFHD$="H"THENHH$="$"+HH$
1650 D$="%ORG  "+HH$
1660 GOSUB3000
2000 W=A:GOSUB800
2010 IFTN$>""ANDTY$="L"THEND$=TN$+":":PRINT#3,D$:IFAG>=8
2020 HH=A                                             THENGOSUB3000
2030 PR$="  "
2040 GOSUB600
2050 PRINT#3,HH$"    ";
2060 GOSUB10
2070 H=II
2080 GOSUB200
2090 PRINT#3,H$;
2100 CD$=CD$(II)
2110 IFCD$=""THENCD$="???"
2120 IFLEN(CD$)=3ORRIGHT$(CD$,2)=" A"THENX$="":PRINT#3,
2130 GOSUB10                                   "        ";:GOTO2540
2140 H=II
2150 GOSUB200
```

```
2160 PRINT#3,H$;
2170 X$=MID$(CD$,4)
2180 CD$=LEFT$(CD$,3)
2190 IFLEFT$(X$,1)="$"THEN2400
2200 CD$=CD$+"   "
2210 IFX$="#"THENCD$=CD$+"#"
2220 IFX$="+"THEN2330
2230 H=II
2240 PR$="$"
2250 GOSUB500
2260 CD$=CD$+H$
2270 IFX$="ZX"THENCD$=CD$+",X"
2280 IFX$="ZY"THENCD$=CD$+",Y"
2290 IFX$="IX"ORX$="IY"THENCD$=LEFT$(CD$,5)+"("+MID$(CD$,6)
2300 IFX$="IX"THENCD$=CD$+",X)"
2310 IFX$="IY"THENCD$=CD$+"),Y"
2320 GOTO2540
2330 HH=II
2340 IFHH>127THENHH=HH-256
2350 HH=HH+A
2360 PR$="$"
2370 GOSUB600
2380 CD$=CD$+HH$
2390 GOTO2540
2400 CD$=CD$+"   "
2410 IFX$="$I"THENCD$=CD$+"("
2420 HH=II
2430 GOSUB10
2440 HH=HH+256*II
2450 PR$="$"
2460 GOSUB600
2470 CD$=CD$+HH$
2480 IFX$="$X"THENCD$=CD$+",X"
2490 IFX$="$Y"THENCD$=CD$+",Y"
2500 IFX$="$I"THENCD$=CD$+")"
2510 H=II
2520 GOSUB200
2530 PRINT#3,H$;
2540 IFLEFT$(X$,1)<>"$"ANDX$>""THENPRINT#3,"   ";
2550 PRINT#3,"    "CD$
2560 IFAG>=8THEND$=CD$:GOSUB3000
2570 IFIS=0THEN2000
2580 IFED$=""THEN1220
2590 IFPATHENPRINT#6,CHR$(0)CHR$(0);
2600 CLOSE15
2610 END
```

Zunächst werden die mnemotechnischen Bezeichnungen aus den DATA-Anweisungen in das Feld CD$() eingelesen und der Fehlerkanal der Floppy geöffnet.

Dann wird am Bildschirm der Dateiname der einzulesenden Symboltabelle erfragt, diese eingelesen und dann wieder zu Eingabe einer Symboldatei gesprungen. Beim Einlesen der Symbole wird der Symboltyp auf ein Zeichen verkürzt und in dem Feld TY$() gespeichert. Wird keine Symboldatei mehr gewünscht, wird die Eingabedatei ED$ erfragt. Ist ED$="", so werden Startadresse und Endadresse des zu disassemblierenden Speicherbereichs erfragt. Wenn die Endadresse EA gleich 0 ist, wird das Programm beendet.

Wurde die Ausgabedatei bereits erfragt (A größer 0), so wird der folgende Teil übersprungen. Wurde eine Eingabedatei gewählt, so wird diese in Zeile 1260 eröffnet und der Disk-Status überprüft.

Ab Zeile 1290 erfolgt die Wahl der Ausgabe. Je nachdem, ob die Nummer des Ausgabegerätes AG kleiners als 8 ist oder nicht, wird einfach eine Datei mit der logischen Nummer 3 eröffnet oder mit der logischen Nummer 6 eine Floppydatei, deren Name AD$ und deren Typ AT$ noch erfragt werden. Ist AT$="P", so wird als Startadresse für das zu erzeugende Programm der Wert 2049 gespeichert. Zusätzlich zur Datei 6 wird noch eine Datei 3 auf dem Bildschirm eröffnet.

Anschließend wird die Variable HD$ besetzt, entweder mit "H" für hexadezimale Ausgabe oder mit "D" für dezimale Ausgabe. Die eingelesenen Symbole werden als Konstantendefinitionen auf der Ausgabedatei (wenn angewählt) gespeichert. Dabei werden nur die Symbole ausgewählt, deren Typkennzeichen TY$() nicht gleich "L" ist. Letztere Symbole werden später als Markendefinitionen gespeichert.

Die Startadresse des zu assemblierenden Bereichs wird aus der Floppydatei gelesen, wenn kein Speicherbereich disassembliert werden soll. Diese Startadresse wird als %ORG-Direktive in eine evtl. angewählte Ausgabedatei geschrieben.

In den Zeilen 2000 bis 2610 befindet sich die Schleife zum fortlaufenden Disassemblieren. Hierbei wird zunächst die aktuelle Adresse ausgegeben. Wenn der Adresse ein gespeicherter Marken-Name entspricht, wird dieser mit einem folgenden Doppelpunkt ausgegeben. Dann wird das nächste Byte geholt und ausgegeben, sowie dessen mnemotechnische Bezeichnung in der Variablen CD$ festgehalten.

festgehalten. Handelt es sich um eine implizite Adressierung (Länge von CD$=3) oder um eine Akkumulatoroperation (die rechten beiden Zeichen von CD$ sind ' A'), so wird zum Abschluß der Schleife bei Zeile 2540 gesprungen. Sonst wird das nächste Byte eingelesen, das zum Operanden der Anweisung gehören muß. Die mnemotechnische Bezeichnung wird nun zerlegt in eine Variable X$, die die Adressierungsart speichert, und in die Variable CD$, die nun neu mit den linken drei Zeichen ihres alten Wertes besetzt wird.

Liegt ein Zwei-Byte-Operand vor (das linke Zeichen von X$ ist "$"), wird zu Zeile 2400 gesprungen. Wenn nicht, handelt es sich um einen Ein-Byte-Operanden, dem dann lediglich je nach Adressierungsart noch einige Zeichen vorne und hinten hinzugefügt werden müssen.

Ein relativer Sprung, der durch X$="+" gekennzeichnet ist, wird ab Zeile 2330 behandelt. Dabei wird dem momentanen Adresswert die Sprungweite hinzuaddiert und dieser Wert als Operand benützt.

Ab Zeile 2400 geschieht die Umwandlung einer Anweisung mit Zwei-Byte-Operanden. Dazu wird zunächst noch ein weiteres Zeichen eingelesen und daraus der vollständige Wert des Operanden gebildet. Der Operand selbst wird dann noch - je nach Adressierungsart - mit entsprechenden Zeichen ergänzt.

In jedem Fall wird ab Zeile 2540 das Ergebnis des Disassemblierungsvorgangs ausgegeben. Die Schleife wird beendet, wenn die Variable IS verschieden von 0 ist. Dann wird wieder zur Angabe einer neuen Start- und Endadresse gesprungen, wenn keine Eingabedatei angewählt wurde. Wurde eine Eingabedatei gewählt, wird der Kommandokanal der Floppy geschlossen und das Programm beendet.

Disassembler **1 - 50260**

Variablen:

Name	Typ	Bereich	Bedeutung
A	G	0...65535	momentane Adresse
AD$	G	Zeichenreihe	Name der Ausgabedatei
AG	G	0...15	Ausgabegerät
AT$	G	"","S","P"	Typ der Ausgabedatei
CD$	H	3 bis 5 Zeichen	Klartext des akt. Bef.
D$	P	Zeichenreihe	Ergebnis einer Zeile
D1$	H	'00' bis '99'	Disk-Status (Spur)
D2$	H	'00' bis '99'	Disk-Status (Sektor)
DS	R	0...99	Disk-Status (Nummer)
DS$	R	Zeichenreihe	Disk-Status (Klartext)
EA	G	0...65535	Endadresse des zu dis-assemblier. Bereichs
ED$	G	Zeichenreihe	Name der Eingabedatei
H	P	0...255	Wert für Hexzahl
H$	R	2 oder 3 Zeichen	Hexadezimalzahl von H
HD$	G	'H','D'	Flag für Hex/Dez
HE$	G	0123456789ABCDEF	für Dez/Hex-Umwand.
HH	P	0...65535	Wert für Hexzahl
HH$	R	4 oder 5 Zeichen	Hexzahl von HH
HI	H	0...255	High-Byte von Zahl
I	H	0...255	Laufvariable
I$	H	1 Zeichen	Nächstes Byte
II	R	0...255	ASC(I$)
IS	R	0,64,2	Eingabestatus
LO	H	0...255	Low-Byte von Zahl
PA	G	2049...65535	Adresse für Ausg.dat.
PR$	G	' ','$'	Präfix vor Hexzahlen
SA	G	0...65535	Startadresse Bereich
TA	G	0...500	Anzahl Symbole
TN$	G	Zeichenreihe	gefundenes Symbol
TY$	G	"U","C","L"	Typ des gef. Symbols
X$	H	0 bis 2 Zeichen	Kürzel f. Adressierung
ZN	G	0...61000	Zeilennummer für Ausgabedatei

Felder (Arrays):

Name	Dimen.	Typ	Bereich	Bedeutung
CD$	0...255	G	3 bis 5 Zeichen	Mnemoteschnische Codes
TV	0...500	G	0...65535	Wert d. Symbole
TN$	0...500	G	bis 8 Zeichen	Symbolnamen
TY$	0...500	G	"U","C","L"	Symboltyp

Dateien :

#	Name	T	Bemerkung
2	ED$	p	zu disassemblierende Datei
6	AD$	p/s	Ausgabedatei für Ergebnis

Unterprogrammaufrufe : (im Hauptprogramm ab 1000)

in	nach	Zweck
1100	900	Disk-Status bestimmen
1270	900	Disk-Status bestimmen
1340	900	Disk-Status bestimmen
1500	600	2-Byte-Operand bilden
1530	3000	Eine Zeile speichern
1560	10	Nächstes Byte holen
1580	10	Nächstes Byte holen
1630	600	2-Byte-Operand bilden
1660	3000	Eine Zeile speichern
2000	800	Symbol suchen
2010	3000	Eine Zeile speichern
2040	600	2-Byte-Operand bilden
2060	10	Nächstes Byte holen
2080	200	2-stellige Hexzahl bilden
2130	10	Nächstes Byte holen
2150	200	2-stellige Hexzahl bilden
2250	500	1-Byte-Operand bilden
2370	600	2-Byte-Operand bilden
2430	10	Nächstes Byte holen
2460	600	2-Byte-Operand bilden
2520	200	2-stellige Hexzahl bilden
2560	3000	Eine Zeile speichern

Verzweigungen nach außen :

in Ze	nach	Bedingung	Bemerkung
1100	END	EA = 0	Eingabe von '0,0'
2610	END	IS ungleich 0	Normales Ende

5.4 Ergänzungen

Für das Disassembler-Programm gilt das gleiche wie das in Kaapitel 3.4 über den Assembler ausgeführte: es gibt noch einige Möglichkeiten, die Leistungsfähigkeit des Programms zu steigern. Zum einen kann man die Geschwindigkeit durch Einbau von Maschinenroutinen erhöhen. Bereits die Routinen, die beim Assembler vorgestellt wurden, können wirksam eingesetzt werden.

Andererseits kann auch der Einsatzbereich des Disassmblers erweitert werden. Die vorliegende Version kann zwar bereits Symbole verwalten, jedoch nur solche, die in einer bereits erstellten Tabelle vorhanden sind. Dies ist auch hilfreich, wenn man z.B. die Namen sämtlicher Basic- und KERNAL-Routinen in einer Datei abgelegt hat.

Manchmal möchte man aber ein besser strukturiertes Listing eines disassemblierten Programms. Man kann dann den Disassembler so erweitern, daß er in einem ersten Durchlauf durch die Objektdatei selbst Symbole für jeden Sprung und jede absolute Adressierung generiert und diese beim zweiten Durchgang einsetzt.

Ein weiterer Mangel ist die ungenügende Kennzeichnung der im Objektprogramm enthaltenen Texte und Konstanten. Mit Hilfe einer Unterscheidung von Sprungadressen und Datenadressen wäre in den meisten Fällen das Problem gelöst.

Man erhält auch bereits eine übersichtlichere Struktur des Listings, wenn man nach jedem unbedingten Sprungbefehl (JMP,RTS,RTI,BRK) eine Leerzeile (bzw. eine Zeile nur mit einem Semikolon) ausgibt.

6

Bedienungsanleitungen

6. Bedienungsanleitungen

In diesem Kapitel sind dei Bedienungsanleitungen für die Programme "ASS4" bzw. "ASS4B" und "DA4" abgebildet. Diese beiden Programme sind auch auf der zum Buch gehörigen Diskette zu finden.

6.1 Assembler

Aufbau eines Quell-Programms

Am Anfang des Quellprogramms muß die Startadresse des zu erstellenden Objekt-Codes definiert werden. Dies geschieht mit einer %ORG oder %INO-Direktive (siehe unten).

Bemerkungen werden einfach durch Semikolon abgetrennt. D.h. der Text nach dem Semikolon wird nicht assembliert, erscheint jedoch auf dem Protokoll.

Das Ende des zu übersetzenden Quellprogramms kann durch eine %END-Direktive gekennzeichnet werden.

Marken (Labels) werden durch einen Doppelpunkt nach dem Markennamen definiert. Nach der Markendefinition können einfache Befehle (keine Direktiven) folgen.

Symbole werden wie folgt definiert:

Symbolname = Ausdruck

Ein **Ausdruck** kann bis zu zwei Terme enthalten, die durch ein '+','-','*' oder '/' verbunden sind.

Ein **Term** kann sein:

- eine positive ganze Zahl
- eine zweistellige oder vierstellige Hexadezimalzahl, gekennzeichnet durch ein $-Zeichen vor der Zahl
- eine achtstellige Binärzahl, ebenfalls durch '$' gekennzeichnet
- $ (aktueller Stand des Programmzählers)
- ein bereits definiertes Symbol
- ein Term mit vorangestelltem Kleinerzeichen (Low-Byte)
- ein Term mit vorangestelltem Größerzeichen (High-Byte)

Mnemonics

Für die mnemotechnischen Bezeichnungen werden die üblichen Abkürzungen verwendet, wie sie auch von MOS-Technology vorgeschlagen werden. Zusätzlich gibt es die Befehle:

BYT	(Bytewert)	reserviert ein Byte
WOR	(Integerwert)	reserviert ein Wort, d.h. 2 Byte, die in der Reihenfolge Low Byte, High Byte kodiert werden.

Erstellung eines Quellprogramms als Programmdatei

Ein Quellprogramm kann wie ein normales Basic-Programm editiert werden. Die dabei verwendeten Zeilennummern haben nur dokumentarischen Charakter, werden aber auf dem Listprotokoll mit ausgegeben. Die Verwendung von einem **Fragezeichen** ist nur in Hochkommas möglich, weil es sonst in einen PRINT-Befehl konvertiert wird. Das Abspeichern des Quellprogramms geschieht mit

SAVE"Programmname.SRC",8

Die angehängten Zeichen ".SRC" sind notwendig, da der Assembler daran das Quellprogramm erkennt.

Erstellen des Quellprogramms als sequentielle Datei

Hierzu wird ein Texteditor benötigt, der das Quellprogramm sequentiell wie einen normalen Text speichert. Es werden keine Zeilennummern benötigt, im Listprotokoll erscheinen deshalb einfach laufende Zeilennummern.

Laden und Starten des Assemblers

Der Assembler wird wie ein normales Basic-Programm mit

LOAD"ASS4",8 (RETURN) bzw. LOAD"ASS4B",8 (RETURN)

geladen und mit RUN gestartet. Nach etwa 10 Sekunden fragt der Rechner nach dem Namen der Quelldatei, der hier ohne den Zusatz ".SRC" eingegeben werden muß. Anschließend fragt der Rechner nach der Ausgabedatei für das Listprotokoll. Hier kann man eingeben:

3 Das Protokoll wird auf dem Bildschirm

ausgegeben
4 oder 5 Das Protokoll wird auf dem entsprechenden Drucker ausgegeben
8 oder 9 Das Protokoll wird auf eine sequentielle Datei "Name.LST" geschrieben

Am Ende des Assemblierungsvorgangs werden vom Rechner die Vorwärtsverweise eingebaut. Stößt der Assembler dabei auf ein noch nicht definiertes Symbol, so wird dieses am Bildschirm erfragt. Die Eingabe kann dann dezimal oder hexadezimal erfolgen.

Zum Abschluß wird noch die Symboltabelle als sequentielle Datei "Programmname.SYM" gespeichert und auf dem Listprotokoll ausgegeben.

Laden des Objektprogramms:

Das Objektprogramm kann mit

```
LOAD"Programmname.OBJ",8,1 : NEW
```

an die im Quellprogramm angegebene Startadresse geladen werden. Näheres siehe Kapitel 2.5.

Direktiven

Jede Direktive beginnt mit einem %-Zeichen als erstes Zeichen einer Programmzeile. Der vorliegende Assembler kennt sechs Direktiven:

%ORG (Start-Adresse)

Mit %ORG wird die Start-Adresse des Objekt-Programms definiert. Die Angabe des Wertes kann hexadezimal oder dezimal erfolgen.

%INO

Durch diese Direktive wird die Start-Adresse während des Assemblierungsvorgangs vom Bildschirm erfragt.

%END

Markiert das Ende des zu assemblierenden Quellprogramms.

%ASC "(einfacher Text)"

Der in Anführungszeichen angegebene Text wird an dieser Stelle in das Objekt-Programm eingebaut. Im Listprotokoll erscheinen aus Platzgründen nur die letzten drei Zeichen in der Spalte Code.

%DUP (Anzahl), (einfache Anweisung)

Mit dieser Anweisung kann man den Objektcode einer Anweisung mehrmals hintereinander in das Objekt-Programm einbauen.

Beispiele:

%DUP16,BYT 0	fügt sechzehn Nullen in das Objekt-Programm ein.
%DUP4,ASL A	verschiebt den Akkumulator um 4 Bits nach links

Im Listprotokoll steht unter der Spalte Code nur der letzte der wiederholten Befehle.

Sowohl der Ausdruck für die Anzahl der Wiederholungen als auch die Ausdrucke in der folgenden einfachen Anweisung können Symbole enthalten. Jedoch müssen diese Symbole zu diesem Zeitpunkt definiert sein.

%INCLUDE "Quelldatei"

Mit diesem Befehl wird die angegebene Quelldatei in das Quellprogramm eingefügt. Wenn die eingefügte Datei zu Ende ist, wird mit der Abarbeitung der originären Quelldatei fortgefahren.

Angabe der Adressierungsarten

Die Adressierungsarten werden weitgehend so angegeben, wie von MOS-Technologie empfohlen, d.h eine Immediate-Adressierung wird durch ein "#" gekennzeichnet.

Es ergeben sich jedoch folgende Besonderheiten: Der Assembler verwendet automatisch die Zero-Page-Adressierung, wenn die Adresse kleiner als 256 ist. Will man aber die 3-Byte-Form des Befehls (also die absolute Adressierung), so

muß man dem Operanden ein Rufzeichen voranstellen. Dies ist manchmal notwendig, wenn man Adressen in der Zero-Page X-indiziert ansprechen möchte, weil die meisten Befehle nur eine absolut X-indizierte Adressierung kennen.

Ist dem Assembler die Größe des Adressoperanden nicht bekannt, so reserviert er grundsätzlich den Platz für eine absolute Adressierung. Wird das fehlende Symbol anschliessend definiert, so kann der Assembler natürlich ein eventuell zuviel reserviertes Byte nicht mehr löschen. Deshalb ist im Assembler die Möglichkeit vorgesehen, durch ein dem Operanden vorgestelltes Klammeraffe-Zeichen die Zero-Page-Adressierung zu erzwingen.

Bei relativen Sprüngen ist es am sinnvollsten, den Operanden als symbolische Marke anzugeben. Wenn man dies nicht tun will, so muß man beachten, daß dann das Sprungziel in der Form $+2 (Sprungweite + 2) angegeben werden muß. Den Code "FO 02" erhält man also durch den Assembler-Befehl "BEQ $+4".

6.2 Disassembler

Der Disassembler wird wie ein normales Basic-Programm mit

LOAD"DA4",8

geladen und mit RUN gestartet. Nach einer kurzen Zeit fragt das Programm nach einer Symboldatei. Hier ist der vollständige Name einer Datei einzugeben, aus der die Symbole gelesen werden sollen, die später im Listing auftauchen. Wird einfach RETURN gedrückt, so wird keine Symboldatei mehr gelesen.

Dann erwartet der Rechner den Namen der Eingabedatei. Hier muß das Objekt-Programm angegeben werden, welches disassembliert werden soll. Wird hier einfach RETURN gedrückt, so fragt das Programm nach dem zu disassemblierenden Adressbereich. Dadurch kann man Programme disassemblieren, die im Moment im Speicher stehen.

Anschließend fragt der Rechner nach dem Ausgabegerät. Hier kann man eingeben:

3 - Ausgabe auf dem Bildschirm
4 - Ausgabe auf dem Drucker
5 - Ausgabe auf dem 2. Drucker
8 - Ausgabe auf eine Floppy-Datei

Wurde '8' eingegeben, muß noch der Name und der Typ der Ausgabedatei durch Komma getrennt eingegeben werden. Als Typ kann man 'S' oder 'P' angeben. In beiden Fällen erhält man eine Datei, die mit dem oben beschriebenen Assembler gelesen werden kann.

Schließlich kann man noch durch einfaches Drücken von 'H' oder 'D' wählen, ob die Operanden hinter den mnemotechnischen Bezeichnungen in hexadezimaler oder dezimaler Form angezeigt werden sollen.

Dann wird das gesamte Programm bzw. der angegebene Adressbereich disassembliert und auf dem Ausgabegerät ausgegeben.

Wurde eine Datei disassembliert, wird das Programm anschließend beendet, sonst fragt der Rechner wieder nach einem Adress-Bereich. Das Programm kann beendet werden, indem hier '0,0' eingegeben wird.

Anhang

Anhang 1

Tabelle der Parametertypen:

- A - Ausgabeparameter von diesem Unterprogramm
- E - Eingabeparameter für dieses Unterprogramm
- G - Globale Variable
- H - Hilfsvariable
- P - Aufrufparameter an Unterprogramm
- R - Rückgabeparameter von Unterprogramm
- T - Transienter Parameter (ist in einem Unterprogramm gleichzeitig Eingabeparameter (E) und Aufrufparameter (P) bzw. A und R

Globale Variablen gibt es zwar bei Basic nicht, da Basic keine block-orientierte Sprache wie z.B. PL/1 ist, aber in diesem Buch ist dieser Begriff für Variablen verwendet worden, die man in anderen Sprachen global definiert hätte.

Da der verwendete Typenraddrucker die Zeichen 'größer als' und 'kleiner als' nicht drucken kann, wurden diese Zeichen wie folgt ersetzt:

NE - Not Equal	/ ungleich
GT - Greater Than	/ größer als
LT - Less Than	/ kleiner als
GE - Greater or Equal	/ größer gleich
LE - Less or Equal	/ kleiner gleich

Die in den Kapiteln angegebenen Zeilenbereiche können mit denen der abgedruckten Listings differieren, weil offensichtliche REM-Zeilen (in der Regel die Überschriften von Unterprogrammen) nicht mitgedruckt wurden.

Anhang 2

In den Programmlistings tauchen folgende Bildschirmsteuer-
zeichen auf:

BILDSCHIRMSTEUERCODES

ZEICHEN	CODE
CURSOR NACH UNTEN	
CURSOR NACH OBEN	
CURSOR NACH RECHTS	
CURSOR NACH LINKS	
DEL (ZEI. LOESCHEN)	
HOME (CRSR OBEN LINKS)	
CLR (BILDSCHIRM LOESCHEN)	
REVERSE EIN	
REVERSE AUS	

Hexadezimal - Dezimal - Umwandlungstabelle

(In der linken Spalte befindet sich das erste Zeichen, in der Kopfzeile das zweite Zeichen. Die Beiden rechten Spalten bilden die Grundlage für größere Werte, z.B. $2000 = 8192)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	00	000
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	256	4096
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	512	8192
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	768	12288
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	1024	16384
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	1280	20480
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	1536	24576
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	1792	28672
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	2048	32768
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	2304	36864
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	2560	40960
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	2816	45056
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	3072	49152
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	3328	53248
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	3584	57344
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	3840	61440

Beispiele: $7F =127 ; $F7 = 247 ; $B00 = 2816 ; $B000 = 45056
$A77 = 2560+119 = 2679 ; $ABCD = 40960+2816+205 = 43981

Anhang: Komplettlisting der gemischten Version

```
0 IFL=0THENL=1:LOAD"ASS.OBJ",8,1
1 GOTO50000
1000 REM #  A S S E M B L I E R E N                          #
1010 FP=2
1020 INPUT"DATEI                    ";F$
1030 FT=2
1040 OPEN2,8,2,F$+".SRC,P"
1050 GOSUB25000
1060 IFDS=0THENSYSP2,2:SYSP2,2:GOTO1130
1070 IFDS<>64THENPRINTDS$:CLOSE2:GOTO1020
1080 CLOSE2
1090 FT=1
1100 OPEN2,8,2,F$+".SRC,S"
1110 GOSUB25000
1120 IFDSTHENPRINTDS$:CLOSE2:GOTO1020
1130 PRINT"■■EINGABEDATEI IST "F$".SRC"
1140 PRINT#15,"S:"+F$+"...."
1150 PRINT#15,"S:"+F$+".UND"
1160 OPEN3,8,1,F$+"....,P,W"
1170 GOSUB25000
1180 IFDSTHENPRINTDS$:STOP
1190 INPUT"■LIST-GERAET (ADR.)    3■■■■";LD
1200 PRINT"■LISTE ZU ";
1210 IFLD<8THENOPEN4,LD:PRINT"GERAET";LD:GOTO1260
1220 PRINT"DATEI "F$".LST AUF GERAETENR."LD
1230 OPEN4,LD,4,"@:"+F$+".LST,S,W"
1240 GOSUB25000
1250 IFDSTHENPRINTDS$:STOP
1260 PRINT#4,"*** COMMODORE 64   6502-ASSEMBLER ***     ";
1270 PRINT#4,"     VERSION 1.5  (09.03.84)"
1280 PRINT#4,"ASSEMBLIEREN   VON "F$".SRC"
1290 PRINT#4,"OBJECT-DATEI   IST "F$".OBJ"
1300 PRINT#4,"SYMBOL-TABELLE IST "F$".SYM"
1310 PRINT#4
1320 PRINT#4,"ZEILE  ADR.  OBJ     * QUELLTEXT"
1330 PRINT#4
1340 GOSUB22000
1350 TT$=T$
1360 GOSUB10000
1370 SYSP5,AD,HH$
1380 PRINT#4,ZN$"  "HH$;"  "LEFT$(C$+"          ",7);U$"  "TT$
1390 EB=PEEK(Q4)
1400 IFEBTHENFORI=1TOEB:PRINT#4,"FEHLER: ";ER$(PEEK(Q4+I)):NEXT
1410 POKEQ4,0
1420 EN=EN+EB
1430 IFC$=""THEN1480
1440 AD=AD+1
1450 PRINT#3,CHR$(USR(LEFT$(C$,2)));
```

```
1460 C$=MID$(C$,3)
1470 GOTO1430
1480 IFIS=0THEN1340
1490 IFIS=64THENPRINT#4,"■DATEIENDE ERREICHT.
1500 PRINT#4
1510 PRINT#4,EN"FEHLER."
1520 IFLD<>3THENPRINTEN"FEHLER."
1530 CLOSE3
1540 CLOSE2
2000 REM #  W E R T E   NACHTRAEGLICH  EINSETZEN         #
2010 PRINT#15,"S:"+F$+".OBJ"
2020 GOSUB25000
2030 IFDS>1THENPRINTDS$:STOP
2040 IFUTHEN2090
2050 PRINT#15,"R:"+F$+".OBJ="+F$+"...."
2060 GOSUB25000
2070 IFDSTHENPRINTDS$:STOP
2080 GOTO2510
2090 OPEN3,8,1,F$+".OBJ,P,W"
2100 GOSUB25000
2110 IFDSTHENPRINTDS$:STOP
2120 OPEN2,8,2,F$+"....,P,R"
2130 U1=0
2140 GOSUB25000
2150 IFDSTHENPRINTDS$:STOP
2160 AD=SA-3:REM STARTADRESSE UEBERLESEN
2170 U1=1:UA=UA(U1):UT=UT(U1):UN$=UN$(U1)
2180 AD=AD+1
2190 SYSP2,2:IS=ST:II=PEEK(Q3)
2200 IFUA>ADORU1>UTHENPRINT#3,CHR$(II);:GOTO2440
2210 SI=0
2220 IFUT=1ORUT=3ORUT=4ORUT=8THENSI=1
2230 IFUT=2ORUT=5ORUT=6ORUT=7THENSI=2
2240 IFSI=2THENI1=II:SYSP2,2:IS=ST:I2=PEEK(Q3)
2250 IFUT=0THENSTOP
2260 T$=LEFT$(T$+SP$,8)
2270 SYSPB,UN$,TV,TY
2280 IFTY=0THENGOSUB21000
2290 IFUT=8THEN2380
2300 A=II-128+TV
2310 IFSI=1AND(A<0ORA>255)THENGOSUB26000
2320 IFSI=1THENPRINT#3,CHR$(A);:GOTO2420
2330 A=I1+256*I2-32768-128+TV
2340 H=INT(A/256)
2350 PRINT#3,CHR$(A-256*H)CHR$(H);
2360 AD=AD+1
2370 GOTO2420
2380 A=II+256*(II>127)+TV-(AD+1)
2390 IFA>127ORA<-128THENGOSUB26000
2400 IFA<0THENA=A+256
2410 PRINT#3,CHR$(A);
```

```
2420 IFU1=UTHENU1=U+1:GOTO2440
2430 U1=U1+1:UA=UA(U1):UT=UT(U1):UN$=UN$(U1)
2440 IFIS=0THEN2180
2450 CLOSE2
2460 CLOSE3
2470 CLOSE6
2480 PRINT#15,"S:"+F$+"...."
2490 GOSUB25000
2500 IFDS>1THENPRINTDS$:STOP
2510 PRINT#4,"FERTIG ASSEMBLIERT."
2520 IFLD<>3THENPRINT"FERTIG ASSEMBLIERT."
2530 GOSUB20000
2540 CLOSE15
2550 CLOSE4
2560 END
10000 REM * EINE ZEILE (T$) ASSEMBLIEREN
10010 C$=""
10020 U$=" "
10030 SYSP1,T$
10040 SYSP9,T$,A,;
10050 IFATHENT$=LEFT$(T$,A-1)
10060 SYSP1,T$
10070 IFLEFT$(T$,1)="%"THENGOSUB13000:RETURN
10080 A$="="
10090 SYSP8,A$,T$,A
10100 IFA=0THEN10220
10110 T0$=LEFT$(T$,A-1)
10120 T$=MID$(T$,A+1)
10130 SYSP1,T$
10140 SI=2
10150 KM=0
10160 GOSUB15000
10170 T$=T0$
10180 SYSP1,T$
10190 T$=LEFT$(T$+SP$,8)
10200 SYSPC,T$,W,1
10210 RETURN
10220 IFAD$=""THENSYSPA,19:AD$=CHR$(0)+CHR$(0)
10230 SYSP9,T$,A,:
10240 IFA=0THEN10300
10250 TN$=LEFT$(T$,A-1)
10260 T$=MID$(T$,A+1)
10270 SYSP1,T$
10280 TN$=LEFT$(TN$+SP$,8)
10290 SYSPC,TN$,AD,2
10300 A$=" "
10310 SYSP8,A$,T$,A
10320 IFA<>4ANDLEN(T$)>3THENSYSPA,1:RETURN
10330 IFT$=""THENRETURN
10340 SYSPD,LEFT$(T$,3):K=PEEK(782)
10350 KM=KM(K)
```

```
10360 IFK=0THENSYSPA,1:RETURN
10370 T$=MID$(T$,4)
10380 GOSUB14000
10390 IFPEEK(Q4)THENRETURN
10400 IFMO>=0ANDKM=0THENSYSPA,7:RETURN
10410 IFMO=-1ANDKM>0THENSYSPA,3:RETURN
10420 ONKM+1GOTO11000,11100,11200,11300,11400,11500
10430 STOP
11000 M=0
11010 GOSUB12000
11020 RETURN
11100 M=2
11110 IFMO=1ORMO=4THENM=0
11120 IFMO=10THENM=1
11130 GOSUB12000
11140 SYSP5,W,HH$
11160 C$=C$+RIGHT$(HH$,2)+LEFT$(HH$,2)
11170 RETURN
11200 M=MO
11210 GOSUB12000
11220 IFMO=9THENRETURN
11230 H1=(MO=0ORMO=1ORMO=2ORMO=3ORMO=7ORMO=8)
11240 IFH1THENSYSP6,W,H$:C$=C$+H$:RETURN
11250 SYSP5,W,HH$
11270 C$=C$+RIGHT$(HH$,2)+LEFT$(HH$,2)
11280 RETURN
11300 IFW>255THENSYSPA,8:RETURN
11310 SYSP6,W,C$
11340 RETURN
11400 SYSP5,W,HH$
11420 C$=RIGHT$(HH$,2)+LEFT$(HH$,2)
11430 RETURN
11500 M=1
11510 IFMO=1ORMO=4THENM=0
11520 GOSUB12000
11530 H=W-AD-2
11540 IFH>127ORH<-128THENSYSPA,17:RETURN
11550 IFH<0THENH=H+256
11560 SYSP6,H,H$
11570 C$=C$+H$
11580 RETURN
12000 REM * C$ MIT CODE BESETZEN
12010 C$=KC$(K,M)
12020 IFC$=""THENSYSPA,6
12030 RETURN
13000 REM * %DIRECTIVEN AUSWERTEN
13020 GOSUB24000
13030 ONDGOTO13100,13200,13300,13400,13500,13600
13040 SYSPA,2
13060 RETURN
13100 REM *** %ORG DIRECTIVE ***
```

```
13105 IFAD$>""THENSYSPA,20:RETURN
13110 T$=MID$(T$,5)
13115 KM=4
13120 SYSP1,T$
13125 GOSUB16000
13130 IFPEEK(Q4)THENRETURN
13135 AD=W
13140 HH=INT(AD/256)
13145 H=AD-256*HH
13150 AD$=CHR$(H)+CHR$(HH)
13155 PRINT#3,AD$;
13160 SA=AD
13165 RETURN
13200 REM *** %END DIRECTIVE ***
13210 IS=320
13220 RETURN
13300 REM *** %INO DIECTIVE ***
13305 IFAD$>""THENSYSPA,20:RETURN
13310 KM=4
13315 INPUT"▯STARTADRESSE    ████";T$
13320 IFT$="  "THEN13315
13325 GOSUB16000
13330 IFPEEK(Q4)THENRETURN
13335 AD=W
13340 HH=INT(AD/256)
13345 H=AD-256*HH
13350 AD$=CHR$(H)+CHR$(HH)
13355 PRINT#3,AD$;
13360 SA=AD
13365 RETURN
13400 REM *** %DUP DIRECTIVE ***
13404 T$=MID$(T$,5)
13408 SYSP9,T$,A,,
13412 IFA=0THENSYSPA,3:RETURN
13416 T3$=MID$(T$,A+1)
13420 T$=LEFT$(T$,A-1)
13424 SI=2
13428 KM=0
13432 GOSUB15000
13436 AN=W
13440 T$=T3$
13444 GOSUB10000
13448 IFAN=0THENC$=""RETURN
13452 IFAN=1THENRETURN
13456 CC$="":A=0
13460 IFA>=LEN(C$)THEN13476
13464 CC$=CC$+CHR$(USR(MID$(C$,A+1,2)))
13468 A=A+2
13472 GOTO13460
13476 FORI=1TOAN-1
13480 PRINT#3,CC$;
```

```
13484 AD=AD+LEN(CC$)
13488 NEXT
13492 RETURN
13500 REM *** %ASC DIRECTIVE ***
13505 T$=MID$(T$,5)
13510 SYSP9,T$,A,"
13515 IFA=0THENSYSPA,3:RETURN
13520 T$=MID$(T$,A+1)
13525 SYSP9,T$,A,"
13530 IFATHENT$=LEFT$(T$,A-1)
13535 C$=""
13540 IFT$=""THENRETURN
13545 SYSP6,ASC(RIGHT$(T$,LEN(T$))),H$
13550 C$=H$+C$
13555 T$=LEFT$(T$,LEN(T$)-1)
13560 IFLEN(C$)<6THEN13540
13565 PRINT#3,T$;
13570 AD=AD+LEN(T$)
13575 RETURN
13600 REM * %INCLUDE-DIRECTIVE                          *
13605 SYSP9,T$,A,"
13610 IFA=0THENSYSPA,24:RETURN
13615 T$=MID$(T$,A+1,17)
13620 SYSP9,T$,A,"
13625 IFA=0THENSYSPA,24:RETURN
13630 T$=LEFT$(T$,A-1)
13635 OPEN6,8,6,T$
13640 GOSUB25000
13645 IFDSTHENSYSPA,23:CLOSE6:RETURN
13650 FP=6
13655 IFFT=2THENSYSP2,FP:SYSP2,FP
13660 RETURN
14000 REM * MODE FESTSTELLEN                            *
14010 REM MODE MO : -1=NO OPER.;  0=# USW. S.ZEILE 54030
14020 REM MODE MO = 10  : INDIRECT
14030 REM SIZE:  SI=1 -> BYTE      SI=2 -> WORD
14040 SYSP1,T$
14050 IFT$=""THENMO=-1:RETURN
14060 IFT$="A"THENMO=9:RETURN
14070 SYSP9,T$,A(0),"#(),.@!
14080 IFA(2)<>-(A(3)>0)THENSYSPA,4:RETURN
14090 L=LEN(T$)
14100 XY=(RIGHT$(T$,1)<>"X"ANDRIGHT$(T$,1)<>"Y")
14110 IFA(4)THENIFA(5)>0ORA(4)<>L-1ORXYTHEN14410
14120 IFA(5)THENIFA(4)>0ORA(5)<>L-1ORXYTHEN14410
14130 IFA(1)>1ORA(6)>1ORA(7)>1THENSYSPA,5:RETURN
14140 IFA(1)THENIFA(2)+A(3)+A(4)+A(5)+A(6)+A(7)THEN14410
14150 IFA(6)THENIFA(7)THEN14410
14160 IFA(7)THENIFA(6)THEN14410
14170 IFA(6)+A(7)THENIFA(1)+A(2)+A(3)THEN14410
14180 IFA(1)THENMO=0:T$=MID$(T$,2):GOTO14440
```

```
14190 IFRIGHT$(T$,3)=",X)"THENMO=7:GOTO14430
14200 IFRIGHT$(T$,3)=".X)"THENMO=7:GOTO14430
14210 IFRIGHT$(T$,3)="),Y"THENMO=8:GOTO14430
14220 IFRIGHT$(T$,3)=").Y"THENMO=8:GOTO14430
14230 IFRIGHT$(T$,1)=")"THENMO=10:T$=MID$(T$,2,L-2):SI=2
14240 ZP=0                                   :GOSUB15000:RETURN
14250 SI=2
14260 IFA(6)THENZP=1:T$=MID$(T$,2):L=L-1:SI=1
14270 IFA(7)THENZP=2:T$=MID$(T$,2):L=L-1
14280 IFRIGHT$(T$,2)=",X"THENMO=5:GOSUB14370:GOTO14340
14290 IFRIGHT$(T$,2)=".X"THENMO=5:GOSUB14370:GOTO14340
14300 IFRIGHT$(T$,2)=",Y"THENMO=6:GOSUB14370:GOTO14340
14310 IFRIGHT$(T$,2)=".Y"THENMO=6:GOSUB14370:GOTO14340
14320 MO=4
14330 GOSUB14380
14340 IFPEEK(Q4)THENRETURN
14350 IFW<256ANDZP<2THENMO=MO-3
14360 RETURN
14370 T$=LEFT$(T$,L-2)
14380 IFZP=1THENMO=MO-3
14390 GOSUB15000
14400 RETURN
14410 SYSPA,10
14420 RETURN
14430 T$=MID$(T$,2,L-4)
14440 SI=1
14450 GOSUB15000
14460 RETURN
15000 REM * DOPPELAUSDRUCK AUSWERTEN
15010 SYSP9,T$,A(0),"#(),.@!+-*/$"
15020 IFA(1)+A(2)+A(3)+A(4)+A(5)+A(6)+A(7)THEN15220
15030 A=-(A(8)>0)-(A(9)>0)-(A(10)>0)-(A(11)>0)
15040 IFA=0THENGOSUB16000:GOTO15200
15050 IFA>1THENSYSPA,12:RETURN
15060 A=A(8)+A(9)+A(10)+A(11)
15070 T1$=LEFT$(T$,A-1)
15080 T2$=MID$(T$,A+1)
15090 T$=T1$
15100 GOSUB16000
15110 W1=W
15120 T$=T2$
15130 GOSUB16000
15140 W2=W
15150 IFPEEK(Q4)THENRETURN
15160 IFA(8)THENW=W1+W2
15170 IFA(9)THENW=W1-W2
15180 IFA(10)THENW=W1*W2
15190 IFA(11)THENW=INT(W1/W2)
15200 IFW>=256↑SITHENSYSPA,13
15210 RETURN
15220 SYSPA,10
```

```
15230 RETURN
16000 REM * EINZELAUSDRUCK AUSWERTEN                             *
16010 SYSP1,T$
16020 HW=0
16030 IFLEFT$(T$,1)="<"THENHW=1:T$=MID$(T$,2)
16040 IFLEFT$(T$,1)=">"THENHW=2:T$=MID$(T$,2)
16050 IFT$="$"THENW=AD:GOTO16510
16060 IFLEFT$(T$,1)="%"THENT$=MID$(T$,2):GOTO16110
16070 IFLEFT$(T$,1)<>"$"THEN16150
16080 T$=MID$(T$,2)
16090 IFLEN(T$)<5THENW=USR(T$):GOTO16510
16100 IFLEN(T$)<>8THENSYSPA,14:RETURN
16110 BB$=T$
16120 GOSUB23000
16130 W=BB
16140 GOTO16510
16150 IFASC(T$)>=48ANDASC(T$)<=57THENW=VAL(T$):GOTO16510
16160 T$=LEFT$(T$+SP$,8)
16170 SYSPB,T$,W,TY
16180 IFTYTHEN16510
16190 UA=AD+1
16200 UN$=T$
16210 ONKMGOTO16250,16300,16370,16400,16430
16220 SYSPA,16
16230 UT=0
16240 GOTO16460
16250 IFMO=1ORMO=4THENUT=7:GOTO16450
16260 IFMO=10THENUT=6:GOTO16450
16270 SYSPA,6
16280 UT=0
16290 GOTO16460
16300 IFMO=0THENUT=1:GOTO16450
16310 IFMO<4THENUT=3:GOTO16450
16320 IFMO<=6THENUT=5:GOTO16450
16330 IFMO<=8THENUT=4:GOTO16450
16340 SYSPA,6
16350 UT=0
16360 GOTO16460
16370 UT=1
16380 UA=AD
16390 GOTO16450
16400 UT=2
16410 UA=AD
16420 GOTO16450
16430 UT=8
16440 GOTO16450
16450 GOSUB19000
16460 U$="R"
16470 W=32768+128:REM DEFAULT WERT FUER ADRESSEN $8080
16480 IFUT=1ORUT=3ORUT=4THENW=128:REM DEFWERT FUER BYTEW.
16490 IFUT=8THENW=AD+2:REM DEFWERT FUER BRANCHES
```

```
16500 REM
16510 IFHW=1THENW=W-256*INT(W/256)
16520 IFHW=2THENW=INT(W/256)
16530 RETURN
19000 REM * WERT IN DATEI DER UNDEF. SYMBOLE EINTRAGEN  *
19010 U=U+1
19020 UA(U)=UA
19030 UT(U)=UT
19040 UN$(U)=UN$
19050 RETURN
20000 REM * SYMBOLTABELLE SPEICHERN UND DRUCKEN          *
20010 PRINT#4
20020 PRINT#4,"SYMBOLE:"
20030 PRINT#4,"NAME      T WERT"
20040 PRINT#4,"----------------"
20050 PRINT#15,"S:"+F$+".SYM"
20060 OPEN5,8,5,F$+".SYM,S,W"
20070 GOSUB25000
20080 IFDSTHENPRINTDS$:STOP
20090 SYSPF,5
20100 CLOSE5
20110 CMD4
20120 SYSPE
20130 PRINT#4
20140 RETURN
21000 REM * MANUELLE EINGABE VON SYMBOLWERTEN            *
21010 PRINTUN$;
21020 INPUTT$
21030 T$=LEFT$(T$+SP$,8)
21040 POKEQ4,0
21050 GOSUB16000
21060 IFPEEK(Q4)THEN21000
21070 IFUT=1THENTY=1:SI=1
21080 IFUT=2THENTY=1:SI=2
21090 IFUT=3ORUT=4THENTY=1:SI=1
21100 IFUT=5ORUT=6THENTY=1:SI=2
21110 IFUT=7ORUT=8THENTY=2:SI=2
21120 IFW>=256↑SITHEN21000
21130 TV=W
21140 SYSPC,UN$,W,TY
21150 RETURN
22000 REM * EINE ZEILE EINLESEN                          *
22010 IFFT=1THENSYSP4,FP,T$:IS=ST:ZN=ZN+1:GOTO22050
22020 SYSP3,FP,T$
22030 IFPEEK(Q2)=0ANDPEEK(Q2+1)=0THENIS=64:RETURN
22040 ZN=PEEK(Q1)+256*PEEK(Q1+1)
22050 ZN$=RIGHT$("      "+STR$(ZN),5)
22060 IFIS>0ANDFP<>2THENFP=2
22070 RETURN
23000 REM * BESTIMMUNG DES WERTES EINER BINAERZAHL       *
23010 BB=0
```

```
23020 FORI=1TOLEN(BB$)
23030 B=ASC(MID$(BB$,I))-48
23040 IFB<0ORB>1THENSYSPA,15:RETURN
23050 BB=2*BB+B
23060 NEXT
23070 RETURN
24000 REM * NUMMER DER DIRECTIVE BESTIMMEN                   *
24010 FORD=1TOD9
24020 SYSP8,D$(D),T$,A
24040 IFA=0THENNEXT
24050 IFD>D9THEND=0
24060 RETURN
25000 REM * DISK-STATUS-WERTE DS UND DS$ BESTIMMEN           *
25010 INPUT#15,DS,D1$,D2$,D3$
25020 DS$=STR$(DS)+","+D1$+","+D2$+","+D3$
25030 RETURN
26000 REM * FEHLERMELDUNG BEIM EINTRAGEN AUSGEBEN            *
26010 SYSP5,AD,A$
26020 PRINT#4,"FEHLER BEI "A$
26030 A=0
26040 RETURN
50000 REM #  P R O G R A M M   -   V O R S P A N N           #
50010 P1=49152:REM BLANKELI
50020 P2=49155:REM GETZEI
50030 P3=49158:REM GETPZ
50040 P4=49161:REM GETTZ
50050 P5=49164:REM HEX4S
50060 P6=49167:REM HEX2S
50070 P7=49170:REM HEXDEZ
50080 P8=49173:REM INDEX
50090 P9=49176:REM SONDZ
50100 PA=49179:REM FEHLREG
50110 PB=49182:REM TABSUCH
50120 PC=49185:REM TABEINF
50130 PD=49188:REM MNEMO
50140 PE=49191:REM TABDRUCK
50150 PF=49194:REM TABSPEI
50160 PI=49197:REM INIT
50170 Q1=49200:REM PZNR
50180 Q2=49202:REM PZVP
50190 Q3=49205:REM ZEICH
50200 Q4=49208:REM FEHLANZ
50210 SYSPI
50220 OPEN15,8,15
50230 DIMAC(5)
50240 FORI=0TO5
50250 READAC(I)
50260 NEXT
50270 K9=58
50280 DIMKM(K9),KC$(K9,10)
50290 FORI=1TOK9
```

```
50300 READKM(I)
50310 IFAC(KM(I))=0THEN50330
50320 FORJ=0TOAC(KM(I))-1:READKC$(I,J):NEXTJ
50330 NEXTI
50340 D9=6
50350 DIMD$(D9)
50360 FORI=1TOD9
50370 READD$(I)
50380 NEXT
50390 E9=24
50400 DIMER$(E9)
50410 FORI=1TOE9
50420 READER$(I)
50430 NEXT
50440 SP$="        "
50450 AN$=CHR$(34)
50460 DIMA(20),UA(200),UT(200),UN$(200)
50470 GOTO1000
54000 REM %  AC(0..5)      ANZAHL MODES                      %
54010 DATA1:REM TYP 0 IMPLIED
54020 DATA2:REM TYP 1 (JUMPS) ABSOLUTE,INDIRECT
54030 DATA10:REM #,Z,ZX,ZY,A,AX,AY,IX,IY,AC  TYP 2
54040 DATA0:REM BYTE
54050 DATA0:REM WORD
54060 DATA1:REM TYP 5 RELATIVE BRANCHES
55000 REM % KM(1..K9),KC$(1..K9,0..AC(KM(I))-1) %
55040 DATA2,69,65,75,,6D,7D,79,61,71,
55050 DATA2,29,25,35,,2D,3D,39,21,31,
55060 DATA2,,06,16,,0E,1E,,,,0A
55070 DATA5,90
55080 DATA5,B0
55090 DATA5,F0
55100 DATA2,,24,,,2C,,,,,
55110 DATA5,30
55120 DATA5,D0
55130 DATA5,10
55140 DATA0,00
55150 DATA5,50
55160 DATA5,70
55170 DATA3
55180 DATA0,18
55190 DATA0,D8
55200 DATA0,58
55210 DATA0,B8
55220 DATA2,C9,C5,D5,,CD,DD,D9,C1,D1,
55230 DATA2,E0,E4,,,EC,,,,,
55240 DATA2,C0,C4,,,CC,,,,,
55250 DATA2,,C6,D6,,CE,DE,,,,
55260 DATA0,CA
55270 DATA0,88
55280 DATA2,49,45,55,,4D,5D,59,41,51,
```

```
55290 DATA2,,E6,F6,,EE,FE,,,,
55300 DATA0,E8
55310 DATA0,C8
55320 DATA1,4C,6C
55330 DATA1,20,
55340 DATA2,A9,A5,B5,,AD,BD,B9,A1,B1,
55350 DATA2,A2,A6,,B6,AE,,BE,,,
55360 DATA2,A0,A4,B4,,AC,BC,,,,
55370 DATA2,,46,56,,4E,5E,,,,4A
55380 DATA0,EA
55390 DATA2,09,05,15,,0D,1D,19,01,11,
55400 DATA0,48
55410 DATA0,08
55420 DATA0,68
55430 DATA0,28
55440 DATA2,,26,36,,2E,3E,,,,2A
55450 DATA2,,66,76,,6E,7E,,,,6A
55460 DATA0,40
55470 DATA0,60
55480 DATA2,E9,E5,F5,,ED,FD,F9,E1,F1,
55490 DATA0,38
55500 DATA0,F8
55510 DATA0,78
55520 DATA2,,85,95,,8D,9D,99,81,91,
55530 DATA2,,86,,96,8E,,,,,
55540 DATA2,,84,94,,8C,,,,,
55550 DATA0,AA
55560 DATA0,A8
55570 DATA0,BA
55580 DATA0,8A
55590 DATA0,9A
55600 DATA0,98
55610 DATA4
56010 REM %  D$(1..D9)          %DIRECTIVEN                  %
56030 DATAORG,END,INO,DUP,ASC,INCLUDE
58010 REM %  ER$(1..E9)      FEHLERMELDUNGEN                 %
58030 DATA"SYNTAX : FALSCHES MNEMONIC"
58040 DATA"SYNTAX : FALSCHE DIRECTIVE"
58050 DATA"SYNTAX : OPERAND FEHLT"
58060 DATA"SYNTAX : KLAMMERN FALSCH GESETZT"
58070 DATA"SYNTAX : MODEZEICHEN NICHT AN ERSTER STELLE"
58080 DATA"ADRESSIERUNGSART HIER NICHT MOEGLICH"
58090 DATA"KEIN OPERAND ERLAUBT"
58100 DATA"OPERAND MUSS EINE KONSTANTE SEIN"
58110 DATA"OPERAND MUSS EINE MARKE (LABEL) SEIN"
58120 DATA"OPERAND FALSCH SPEZIFIZIERT"
58130 DATA"MARKE ODER KONSTANTE SCHON DEFINIERT"
58140 DATA"AUSDRUCK DARF NUR 2 ARGUMENTE ENTHALTEN"
58150 DATA"WERT ZU GROSS"
58160 DATA"FALSCHE LAENGE EINER HEX- ODER BINAER-ZAHL"
58170 DATA"UNGUELTIGES ZEICHEN IN BINAERZAHL"
```

```
58180 DATA"DIESER WERT MUSS BEREITS HIER DEFINIERT SEIN"
58190 DATA"SPRUNG ZU WEIT"
58200 DATA"ZU VIELE FEHLER"
58210 DATA"STARTADRESSE NICHT DEFINIERT"
58220 DATA"STARTADRESSE BEREITS DEFINIERT"
58230 DATA"UNGUELTIGE HEXADEZIMALZAHL"
58240 DATA"ZU VIELE SYMBOLE"
58250 DATA"DATEI NICHT GEFUNDEN"
58260 DATA"UNGUELTIGER DATEINAME"
```

Die anderen Bände der Serie:

Band 1: Leitfaden für den Erstanwender

Die grundlegenden Begriffe der Programmierung und die Eigenschaften des Commodore 64 werden an einprägsamen Beispielen dargestellt: Sprites und hochauflösende Grafik. Mit Maschinenprogrammen für Grafikbefehle und Assembler.

Band 2: Basic-Spiele

Denkspiele, Wirtschaftsspiele, Kartenspiele, Glücksspiele, Notizblock für Skat, Canasta und Doppelkopf, Biorhythmus

Band 3: Leitfaden für Fortgeschrittene

Multi-Color-Sprites, Multi-Color-Grafik, Sound-Generator, Disassembler, Datenverwaltung mit der Floppy, Deutsche Fehlermeldungen

Band 5: Simon's-Basic

Nützliches im Umgang mit Simon's Basic. Ein erweitertes und Handbuch mit kommentiertem Assembler-Listing

Band 6: Spiele

Nochmals Spiele. Diesmal mit Maschinenprogrammen und Musik, sowie vielen Kniffen.

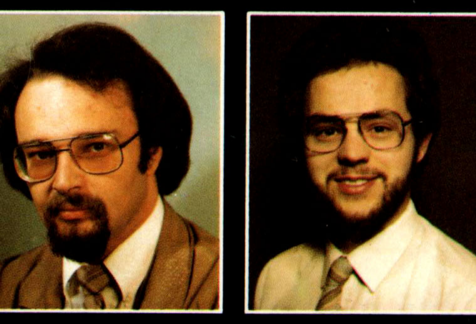
Band 7: Leitfaden für Profis

Wissenswertes zum Umgang mit Light-Pen, Paddles, Joystick; Sound mit HGR, Relative Dateien auf Floppy

Das Commodore 64-Buch

Band 4: Ein Leitfaden für Systemprogrammierer

Das vorliegende Buch wendet sich an diejenigen Leser, welche die Möglichkeiten ihres Commodore 64 mit dem Hilfsmittel der Maschinensprache weiter ausschöpfen wollen.
Das Buch beinhaltet eine Einführung in die Maschinenprogrammierung und eine Anleitung, wie die Maschinenprogramme in Basic-Programme eingebaut werden können. Insbesondere sind ein Assembler in zwei Versionen und ein Disassembler beschrieben. Zum Verständnis dieser Programme ist die Kenntnis von Basic erforderlich.
In zahlreichen Tabellen und Beispielen wird die Wirkungsweise des Prozessors und der Assembler-Programmierung deutlich. Die vorgestellten Programme sind einerseits ein Hilfsmittel für den Programmierer, andererseits werden auch Möglichkeiten aufgezeigt, Programmstücke für eigene Programme zu nutzen. Weitere Beispiele für Maschinenprogramme finden sich auch in anderen Bänden dieser Buchreihe.
Band 1 soll eine Unterstützung für den Erstanwender sein. Die Möglichkeiten des Commodore 64 werden von Beginn an erklärt. Band 2 befaßt sich mit Spielen, aber nicht nur 'Diskette rein — spielen', sondern 'spielend Programmieren lernen' ist das Ziel. Band 3 ist ein Leitfaden für Fortgeschrittene und behandelt Themen, die in Band 1 nicht oder nicht ausführlich behandelt wurden. Band 5 ist einer weitverbreiteten Software-Erweiterung des Commodore 64 gewidmet: dem Simon's Basic. Band 6 bringt schließlich wieder Spiele für Spieleprofis.
Zusätzlich sind zu jedem Band Disketten erhältlich, die dem Leser das Eintippen der beschriebenen Beispiele ersparen.



HANS LORENZ SCHNEIDER geboren am 15.10.53 in Köln. Nach dem Abitur 1973 studierte er von 1976 bis 1980 Informatik an der Bundeswehrhochschule in München. Seit 1980 ist Schneider Inhaber und Geschäftsführer eines Software-Hauses, das sich hauptsächlich mit der Erstellung von Individual-Software für Mikrocomputer befaßt.

WERNER EBERL geboren am 23.2.1962 in München, begann gleich nach dem Abitur 1980 sein Physik-Studium. Seine große Leidenschaft waren schon immer die Computer. Seit 1980 ist er auch als freier Mitarbeiter für ein Software-Haus tätig, wo er für die Umsetzung von Konzepten in lauffähige Programme verantwortlich ist.

Markt&Technik Verlag Aktiengesellschaft
ISBN 3-922120-70-9